Organizaciones autónomas descentralizadas (DAOs) para economía colaborativa utilizando Blockchain

Diego González Blanco
Adrián Guevara Carpizo
Manuel Antonio Fernández Alonso

FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo de fin de grado del Grado en Ingeniería Informática, Facultad de Informática, Universidad Complutense de Madrid

Madrid, Junio 2019

Directores:

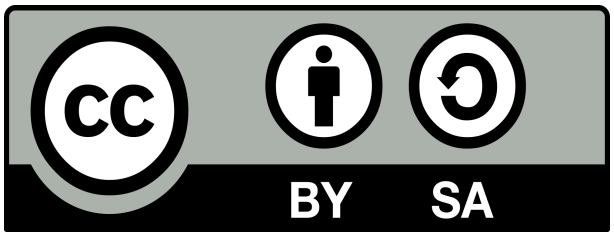
Samer Hassan Collado
David Llop Vila
Juan José Adroher

Autorización de difusión

Diego González Blanco
Adrián Guevara Carpizo
Manuel Antonio Fernández Alonso

Junio, 2019

Los abajo firmantes, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo de Fin de Grado: "Organizaciones autónomas descentralizadas (DAOs) para economía colaborativa utilizando Blockchain", realizado durante el curso académico 2018-2019 bajo la dirección de Samer Hassan Collado, David Llop Vila y Juan José Adroher y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.



Índice general

Agradecimientos

Agradecer especialmente a David Llop Vila su dedicación al proyecto, su alta disponibilidad, su paciencia y sobre todo, los conocimientos que nos ha aportado sobre la materia y las tecnologías que hemos utilizado. Por otra parte, queremos agradecer a Juan José Adroher su aporte a este trabajo por ser la voz de la experiencia y ayudarnos a organizarnos como equipo, participar en el diseño y objetivo de la aplicación y por sus correcciones en la memoria. También agradecer a Antonio Tenorio sus consejos y ayudas para la correcta realización de la memoria. Por último, agradecer a Sammer Hassan el habernos dado la posibilidad de participar en este proyecto y del cual esperamos su pronta recuperación.

Resumen

Este trabajo consiste en la creación de un **modelo de Organización Autónoma Descentralizada (DAO)** sobre Blockchain para sociedades **con un estilo de economía colaborativa**.

A diferencia de otras aplicaciones nuestro trabajo está realizado de manera que, no sólo se pueda aplicar a un tipo de organización como el que hemos hecho de ejemplo (Guerrilla developers) sino que, está hecho para que sirva de **modelo para cualquier organización que tenga un sistema de roles** en los que se necesite hacer ciertos méritos para poder ascender en la organización.

La aplicación está **desarrollada sobre Aragon**, que es un framework que trabaja sobre Ethereum. El uso de Aragon **facilita la creación de estas organizaciones** ya que cuenta con funcionalidades que son de gran ayuda como sistema de votación o gestión de tokens.

Ethereum es una plataforma digital que **utiliza** la tecnología de **cadena de bloques (blockchain)** desarrollada para Bitcoin y expande su uso a distintos tipos de aplicaciones. El uso de este tipo de tecnología **evita** la necesidad de que todo el sistema de gestión este **centralizado** y que se deposite la confianza **en una sola autoridad**.

Palabras clave

Blockchain, DAO, Ethereum, Solidity, Aragon, Smart Contracts, Perfiles, Transacciones, Transiciones, Permisos, React, Redux

Abstract

This work consists of the creation of a **Decentralized Autonomous Organization (DAO) model** on Blockchain for societies **with a style of collaborative economy**.

In contrast to other applications, our work is realized in such a way that, not only can it be applied to a type of organization like the one we have made as an example (Guerrilla developers), but it is made to serve as a **model for any organization that has a system of roles** in which certain roles need to be done in order to be able to ascend in the organization.

The application is **developed on Aragon** which is a framework that works on Ethereum. The use of Aragon **facilitates the creation of these organizations** as it has features that are very helpful as voting system or token management.

Ethereum is a digital platform that **uses blockchain** technology developed for Bitcoin and expands its use to different types of applications. The use of this type of technology **avoids** the need for the entire management system to be **centralized** and for trust to be placed **in a single authority**.

Keywords

Blockchain, DAO, Etherum, Solidity, Aragon, Contracts, Roles, Transactions, Permissions

Capítulo 1

Introducción

Para poder empezar primero, necesitamos tener un contexto que nos permita entender la necesidad de la creación de DAOs basadas en Blockchain. En la mayoría de casos cuando accedemos a servicios a través de internet **estos están centralizados**, es decir, necesitamos acceder a un servidor determinado regido por una empresa determinada. Esto lleva implícito que estamos depositando nuestra confianza en un sistema que no sabemos como funciona y como trata los datos que nosotros introducimos. Además, **estos datos pueden verse comprometidos** si un servidor sufre un ataque o existe cualquier otro problema en la seguridad del mismo. Estos dos aspectos son muy importantes cuando estamos hablando de datos personales, cuentas bancarias...

En 2009 aparece la tecnología de Blockchain como método de seguridad para el Bitcoin. **La Blockchain proponía un sistema distribuido y seguro** utilizando a cada usuario como nodo de un sistema que carecía de un servidor centralizado. Unos años después, en 2015, Ethereum supo aprovechar esta tecnología para la creación de aplicaciones distribuidas.

Gracias al desarrollo de esta tecnología, aparecieron **nuevos conceptos y modelos** de organización como las **DAO**[15] que son organizaciones que están **regidas** a través de **contratos inteligentes (smart contracts)**. Estos contratos son fragmentos de código capaces de ejecutarse de manera autónoma y automática.

Este tipo de organizaciones y la **gestión** de las mismas son muy interesantes ya que pueden suponer un **gran avance en modelos de economía colaborativa DAO**[16][17] en los que los principales problemas que existen son la confianza entre los miembros y la seguridad. El problema radica en que cada organización tiene una jerarquía y **no existe un modelo de DAO** que pueda aplicarse a todas.

En este trabajo se desarrolla **un sistema de perfiles que sirva como modelo para la gestión de usuarios en las DAOs con modelos de economía colaborativa** solucionando así uno de los grandes problemas de este tipo de organizaciones.

1.1. Motivaciones

Sabiendo que uno de los problemas de los modelos de economía colaborativa es **la gestión de usuarios**, como determinar su grado de implicación en la organización y la confianza que se tiene que depositar en cada uno de ellos. El desarrollo de una DAO que permita que **mediante diferentes parámetros** se pueda **determinar en que estado está cada miembro y su puesto en la organización** y, además se pueda **adaptar con facilidad a cualquier tipo de jerarquía** puede suponer un gran avance para este tipo de organizaciones.

1.2. Objetivos

Los objetivos de este trabajo son:

- **Probar y comparar** diferentes frameworks que nos permitan crear DAOs.
- **Crear un modelo** para la creación de DAOs de una manera rápida.
- **Fomentar el uso de DAOs** como solución a diferentes tipos de organizaciones.
- **Fomentar el uso de *software* libre** mediante el desarrollo de una aplicación en Aragon.

Capítulo 2

Introduction

To be in the beginning we need to have a context that allows us to understand the need for the creation of DAOs based on Blockchain. In most cases when we access **services** through the Internet, they **are centralized**, that is, we need to access a specific server governed by a specific company. This implies that we are trusting a system that we do not know how it works and how it processes the data we introduce. In addition, **these data can be compromised** if a server suffers an attack or there is any other problem in the security of it. These two aspects are very important when we are talking about personal data, bank accounts...

In 2009 Blockchain technology appears as a security method for Bitcoin. **Blockchain proposed a distributed and secure system** using each user as a node of a system that had not a centralized server. A few years later, in 2015, Ethereum took advantage of this technology to create distributed applications.

Thanks to the development of this technology, **new concepts and organizational models** appeared, such as **DAOs**, which are organizations **governed by smart contracts**. These contracts are fragments of code that can be executed autonomously and automatically.

This type of organizations and their **management** are very interesting because they can represent a **great advance in models of collaborative economy** in which the main

problems that exist are trust between members and security. The problem is that each organization has a hierarchy and **there is no one DAO model** that can be applied to all.

This study develops a profile system that serves as a **model for user management in DAOs with collaborative economy models**, resolving one of the major problems of this type of organization.

2.1. Incentives

We know that one of the problems of collaborative economy models is the **management of users**, how to determine their degree of involvement in the organization and the trust that has to be placed in each of them. The development of a DAO which, by means of **different parameters, can determine the status of each member** and their position in the organization, and which **can be easily adapted to any type of hierarchy**, can be a great advance for this type of organization.

2.2. Goals

The objectives of this work are:

- **Test and compare** different frameworks that allow us to create DAOs.
- **Create a model** for the creation of DAOs in a fast way.
- **Promote the use of DAOs** as a solution to different types of organizations.
- **Promote the use of free software** through the development of an application in Aragon.

Capítulo 3

Estado del Arte

3.1. Blockchain

La **cadena de bloques** o **blockchain**[1] "*es una estructura de datos en la que la información contenida se agrupa en conjuntos (bloques) a los que se les añade metainformaciones relativas a otro bloque de la cadena anterior en una línea temporal, de manera que gracias a técnicas criptográficas, la información contenida en un bloque solo puede ser repudiada o editada modificando todos los bloques posteriores. Esta propiedad permite su aplicación en entorno distribuido de manera que la estructura de datos blockchain puede ejercer de base de datos pública no relacional que contenga un histórico irrefutable de información*"

Las aplicaciones de esta tecnología son muy variadas, descubriéndose cada vez nuevos campos donde puede emplearse:

- Las **criptomonedas** es quizá la aplicación más conocida del blockchain donde se emplea como un **sistema de transacciones perdurable e inalterable a lo largo del tiempo** para llevar el registro de todos los movimientos realizados con las monedas. Ejemplos representativos pueden ser: Bitcoin, Ethereum, Dogecoin, Peercoin, entre otras.

- **Servicios financieros** que buscan **eliminar intermediarios y procesos burocráticos** que entorpecen, dificultan y ralentizan el proceso en general. Se busca dar

soluciones a actividades como la gestión de activos, seguros, pagos o cualquier sistema que requiera el registro seguro e inalterable de una transacción o dato.

Blockchain hace uso, como su propio nombre indica, de una estructura de datos denominada cadena de bloques que consiste en una lista de bloques enlazada por medio de *hashes* (otras veces llamados punteros *hash*). Cada bloque referencia al bloque anterior el cual se denomina el bloque padre dentro de la cabecera del bloque actual así, la cadena se extiende hasta llegar al primer bloque llamado bloque génesis.

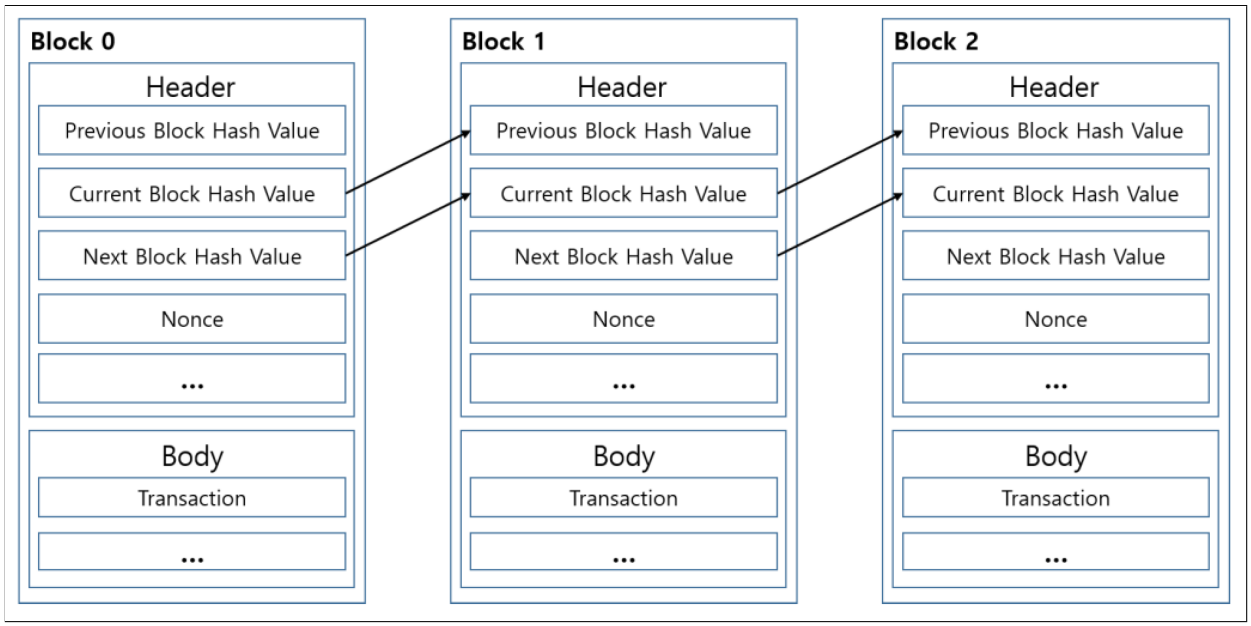


Figura 3.1: *Ejemplo de estructura de bloques dentro de Blockchain*

Los ***hashes*** que se pueden encontrar en cada bloque ofrecen no solo una forma de **encontrar el bloque anterior** sino también una forma de **verificar la integridad** de dicho bloque, permitiendo comprobar si este ha sido modificado o alterado. Por ejemplo, consideremos un individuo que altera la información contenida en el bloque k, si comparamos el *hash* de este nuevo bloque con el *hash* almacenado en el bloque k + 1 veremos que no coinciden, de forma que podemos darnos cuenta de que la cadena ha sido modificada. Solo necesitaríamos el último *hash* de toda la cadena para realizar un seguimiento de la misma

ya que, incluso si el individuo pudiera cambiar todos los bloques de la cadena podríamos comprobar que la integridad se ha visto comprometida utilizando este último *hash*.

3.2. Ethereum

Ethereum[2] fue lanzado en 2015, está basado en Bitcoin y se ha convertido en una de las principales blockchains del mundo. Posee una moneda propia llamada *ether* la cual es puramente digital y puede ser transferida. Una de las características más importantes de esta tecnología es que tanto los datos como el código están **descentralizados y distribuidos** por lo que ningún gobierno ni empresa puede controlarlos.

Ethereum tiene su propio lenguaje de programación lo que permite crear aplicaciones descentralizadas. Estas aplicaciones obtienen todos los beneficios de las criptomonedas y de la blockchain como pueden ser la transparencia y fiabilidad.

El código desarrollado en Ethereum se ejecuta sobre Ethereum Virtual Machine (EVM). Esta EVM se considera una "Máquina de Turing Universal" lo que quiere decir que es lo suficientemente hábil para poder desarrollar cualquier tipo de código definido por un desarrollador. Que permita código supone que requiere de capacidad computacional, es decir, es capaz de ejecutar bucles, operaciones aritmeticológicas, etc. La forma que tiene Ethereum para cuantificar el gasto computacional de las transacciones se denomina *gas*. De no existir el *gas*, nada podría evitar que el sistema se colapsase por completo.

La forma en la que un usuario puede identificarse en la red de Ethereum es mediante las cuentas. Las cuentas están codificadas mediante un conjunto de caracteres en hexadecimal con una longitud de 32 bytes y se pueden asignar a una persona física o a un *smart contract*.

Estas cuentas se utilizan para las transacciones, es decir, mediante esas direcciones en hexadecimal podemos saber la cantidad de ether que posee la cuenta (aclarar que fijar una transacción a la blockchain conlleva un gasto computacional (cobrado en gas) el cuál es

retirado de la cuenta que realiza dicha transacción). Si el Ether de la cuenta es suficiente para realizar la transacción, y cumple con los requisitos criptográficos, dicha transacción se ejecutará y quedará fijada a la blockchain de Ethereum.

Pero lo realmente fundamental de Ethereum, siendo su principal diferencia con otras blockchains, es que transciende el concepto de criptmoneda, dando lugar a los tokens, los cuales están perfectamente definidos en *the business blockchain*, del autor William Mougayar, donde se describen como "una unidad de valor que una organización crea para gobernar su modelo de negocio y dar más poder a sus usuarios para interactuar con sus productos, al tiempo que facilita la distribución y reparto de beneficios entre todos sus accionistas" y dichos tokens están controlados por *smart contracts*.

Un *smart contract* se define como "scripts o pequeños programas, cuyo efecto sea que, una vez concluido un acuerdo y señalados uno o varios eventos desencadenantes, la producción de los eventos programados conlleve la ejecución automática del resto del contrato, sin que quepa modificación, bloqueo o inejecución de la prestación debida"[3] y son fundamentales en las DAOs ya que en ellos se establecen los estatutos de la organización.

3.3. Organizaciones Autónomas Descentralizadas

Se define DAO como "*una organización que* ***se rige por reglas codificadas como programas informáticos*** *denominados "contratos inteligentes", en los que el registro de las transacciones financieras del DAO y las reglas programadas se mantienen en una cadena de bloques, lo que ostensiblemente aumenta drásticamente la transparencia a expensas de la seguridad*" [24]. También se puede interpretar como la **forma más compleja de un contrato inteligente**, donde los estatutos de dicha organización están implementados en el *smart contract*, utilizando complejas reglas de gobernanza.

La primera organización autónoma que conocemos fue Bitcoin ya que **se coordinaba**

únicamente a través de un protocolo de consenso distribuido, que cualquiera podía adoptar. Actualmente, las DAOs están evolucionando muy deprisa, a la par que el software que se necesita para programarlas.

3.3.1. ¿Cómo funciona una DAO?

Como hemos dicho antes, las DAOs se rigen por normas codificadas en programas informáticos llamados *smart contracs*. Dichos contratos están codificados en la Blockchain y **existen de manera autónoma** pero también pueden requerir de una interacción externa (humana, oracles) para realizar cierto tipo de trabajos que el contrato no puede completar por si mismo. A continuación hablaremos de los elementos más importantes que definen una DAO:

Tokens: para que una DAO pueda existir, **necesita algún bien que tenga algún tipo de valor** dentro de la empresa y con el que se pueda **recompensar** a las personas que trabajan en ella.

Autonomía: una vez que se lanza, la organización es **independiente y no puede ser modificada**. Las DAOs son de **código abierto** y por ende, transparentes e incorruptibles. Los *smart contracts* y el registro de transacciones dentro de una DAO está registrado en la Blockchain. **Esto permite que no sea necesario un tercero de confianza** para comprobar que se están realizando las transacciones correctamente.

Los siguientes tres puntos no son fundamentales para el funcionamiento de una DAO, pero debido a que existen una gran cantidad de DAOs que los utilizan, hemos decidido incluirlos:

Empleados: una DAO no puede escribir código, los empleados serán los que **cuiden y desarrollen código** y, además serán también los que interaccionen con el *smart contract* .

Dichos empleados **son elegidos por votación** dentro de la DAO.

Propuestas: es la manera que tiene un miembro de poder **proponer cambios para la organización**. Los mismos miembros de la organización serán los encargados de tomar la decisión sobre estas propuestas.

Votación: una vez presentada una propuesta, se procede a votar. **Los tokens serán necesarios para estas votaciones** y la cantidad que un usuario posea, puede ser relevante en el peso de la votación.

3.3.2. Problemas con las DAOs

Los tres principales conflictos que se han detectado desde la puesta en marcha de las primeras DAOs son:

Condición jurídica no definida: aún no se sabe muy bien donde colocar a una DAO en el ámbito legal ya que, por ejemplo, aún sigue existiendo el debate de si clasificar el Bitcoin como una moneda o una mercancía. La mayoría de proyectos que utilizan blockchain **no tiene un ámbito legal definido**.

Poca participación en las votaciones: el principal monto de personas que participan en una DAO no entienden el valor de sus tokens y la influencia que pueden tener en las votaciones de una DAO.

Complicaciones para corregir código: como ya comentamos antes, para poder solucionar un problema en el código de los contratos, es necesario que **la mayoría de personas** que participen en la DAO dén su consentimiento y esto podría generar un problema de seguridad ya que favoreceria a aquellos que buscan el lucro propio, en detrimento de otros, que desean explotar dicha brecha debido a que habría que generar un nuevo *smart contract*, conseguir que los participantes de la DAO lo aceptasen y migrar todos los fondos a la

nueva DAO constituida. Por otra parte también podría **ralentizar el ritmo de mejora o adaptabilidad** de la DAO. Además, ciertos contratos que se suben a la blockchain no tienen ninguna herramienta programada para corregir el código de los contratos. Aquí entra uno de los casos más importantes en la red de Ethereum que fue el de los "Parity Frozen Funds"[25]. Parity, era una *wallet* que **permitía almacenar ether**. Fue creada por el cofundador de Ethereum, Gavin Woods, y se hizo muy popular porque estaba implementada para **funcionar en diversos sistemas operativos** (Ubuntu, OSX, Docker, Windows...). En noviembre de 2017, un usuario encontró una vulnerabilidad en el código de Parity, del que dependían una gran cantidad de usuarios (no todos, solamente a los que utilizaron esa versión de la wallet) y **congeló un total de 280 millones de dolares en ether**, ya que impidió que la gente pudiese retirar el saldo de sus cuentas. Esto generó un gran descontento en la comunidad de *Parity* y la única solución que se planteó fue un ***hard fork*** de la red de Ethereum. La votación fracasó y los usuarios afectados llevaron el caso ante la justicia estadounidense la cual les dio la razón y obligó a hacer un *hard fork* (Una bifurcación de la red de ethereum, la cual **obliga a todos los usuarios** que quieran **seguir utilizando dicha red**, a tomar esa bifurcación de la cadena de bloques) de la blockchain. En este momento se está llevando acabo este proceso y hay un dilema ético de **¿hasta qué punto está descentralizada la blockchain de Ethereum?** (Ya que si Vitalik Buterin, el creador de Ethereum, toma la decisión de realizar el hard fork, **sin el consentimiento** de la comunidad, **ya no sería una red descentralizada**) ¿Qué pasaría si Vitalik Buterin **se negara a cumplirlo**? son preguntas que aun están sin resolver, pero que pronto tendrán una respuesta. Otro gran problema con las DAOs viene dado de la mano de "The DAO", el cual está explicado en el punto siguiente.

3.3.3. Ejemplos de DAO

El ejemplo más sonado en el mundo de las DAOs fue *"The DAO"*[26]. Fue creada por un grupo de desarrolladores liderado por Christoph Jentzsch cuyo objetivo era **proporcionar**

un nuevo modelo de negocio para la organización de empresas, en concreto, para una empresa de **fondos de inversión** en la que entre toda la comunidad se decidía donde invertir los recursos que poseía la DAO. Fue lanzada en la red de Ethereum y el código de los *smarts contracts* era abierto. Dicha organización n**o estaba vinculada a ningún país o nación** en particular.

Pero dicha DAO tuvo un problema muy importante. Un usuario de la misma se dio cuenta de una vulnerabilidad en el código de los contratos inteligentes, lo cual le permitió trasladar **un tercio del patrimonio** de la DAO a sus cuentas ante la mirada atónita del resto de usuarios que veían como su dinero desaparecía **sin posibilidad de hacer nada**. Los 11.000 participantes de la misma, cuando confiaron en el proyecto *The DAO*, estaban "firmando" un contrato, es decir, aceptando el código programado en el *smart contract* y por lo tanto, este código, se consideraba como **la ley**. El usuario que explotó la vulnerabilidad, se amparó en que sus actuaciones estaban permitidas ya que el sólo estaba utilizando **el código del contrato** que todos los participantes del proyecto habían **aceptado**.

Al final, esto generó un problema en toda la red de Ethereum, (no solo en la comunidad de *The DAO*) llevándose a cabo **una votación para devolver la red al estado anterior** al fraude. La votación se produjo y el **hard fork se realizó** lo cual permitió la solución del problema (este ha sido unos de los *hard fork's* -una vuelta a un estado anterior a la vulnerabilidad de la red de Ethereum- más trascendentales ya que una de las grandes tesitura que conlleva hacer esto es **la posible perdida del dinero** de las personas que hayan invertido en el periodo entre la vulnerabilidad y la realización del *hard fork*).

Por otra parte, incluimos dos ejemplos de criptomonedas debido a ser el token de DAOs con bastante peso y que nos parecen interesantes:

MakerDAO[22]: una criptodivisa que está respaldada por activos de valor. Su principal ventaja es **su inmunidad a la alta volatilidad del mercado** ya que posee sistemas

de emergencia en caso de que esto ocurra. Es transparente debido a estar implementada en blockchain y además permite ver a cualquier persona que lo desee los activos que la respaldan. Dada la baja volatilidad que ofrece, es **muy buena para préstamos**.

Digix Global[23]: **estándar de oro en activos digitales p2p**. Cada token Digix Gold representa 1 gramo de oro en el estándar LMBA, se puede guardar con seguridad en **bóvedas de *Safehouse*** (que puede contener más de 600 toneladas métricas de plata y 30 toneladas de oro y platino) y está diseñado para proporcionar el **almacenamiento más seguro** de criptomoneda en el mercado.

3.4. Framework de implementación de DAOs

La implementación de una DAO se puede realizar desde cero con Solidity, web3 y alguna API de *frontend* como pueden ser Angular o React; también se puede utilizar alguno de los *frameworks* existentes, que ofrecen una estructura que **facilita la implementación de la misma**. A continuación mencionamos los tres *frameworks* más avanzados y relevantes hasta ahora que son: DAOstack, Colony y Aragon.

3.4.1. DAOstack

Ellos lo definen como: "Una plataforma creada para la **Gobernanza Descentralizada** que permite a los colectivos la **auto-organización bajo objetivos y valores comunes**, de manera **sencilla y eficiente**. A DAOstack[4] se le suele denominar como un sistema operativo para la inteligencia colectiva o el Wordpress de las DAOs."

Ha sido lanzada al público en mayo 2018 e incluye una serie de *smarts contracts*, un entorno para desarrollar en JS y una intuitiva interfaz de usuario llamada *Alchemy* que permite a cualquier persona participar en la DAO **sin tener ningún tipo de conocimiento previo**.

También posee una API propia llamada *Arc.js*. Todo el sistema de implementación de Dapps (Aplicaciones descentralizadas de código abierto basadas en blockchain) está basado en esta api y recibe el nombre de Arc, el cual ellos mismos lo definen como "**un entorno revolucionario para la gobernanza escalable**".

Arc posee tres grandes características:

- **Adaptabilidad**: al ser de **código abierto**, DAOstack irá evolucionando **gracias a su comunidad** y permitirá la creación de infinitas Dapps para la gobernanza .
- **Modularidad:** cada DAO que se cree con Arc, estará **compuesta por distintos bloques** (módulos) que se podrán **combinar de maneras muy diversas**. Los módulos permitirán ahorrar en almacenamiento, dinero en operaciones y aumentarán la seguridad.
- **Actualizabilidad:** la estructura con la que se gobiernan las DAOs permite su **actualización de manera sencilla a nuevos esquemas de gobernanza**.

DAOstack posee un token propio llamado *GEN* el cual viene de la primera DAO que se fundó en el ecosistema de DAOstack llamada "Génesis DAO". Dicho token sería el equivalente al *gas* de Ethereum. Su utilizad principal sería la de **impulsar u ocultar propuestas dentro de de la DAO**. Además, también poseen un sistema de reputación la cual es intransferible y te permite votar a favor o en contra de las propuestas.

Para finalizar, DAOstack tiene marcada una hoja de ruta, en la que podemos ver una serie de objetivos futuros marcados como por ejemplo tener para el 3 trimestre de 2019 una versión móvil de *Alchemy* y que, además, DAOstack se pueda integrar con aplicaciones externas como Github, Telegram...

3.4.2. Colony

La idea que hay detrás de Colony[5] es la creación de una organización descentralizada en la cual **el principal valor de la misma sea el mérito**, la reputación y que sea única y exclusivamente digital, sin necesidad de trámites ni gestiones.

Una colonia (colony) se define como un conjunto de *smarts contracts*, que utilizan la blockchain de Ethereum. Este tipo de organizaciones pueden realizar la misma función que cualquier empresa convencional y además **añadir nuevas características que sólo existen gracias a ser descentralizada**. Algunas de las funcionalidades que se pueden hacer en una colonia son: Votaciones, sistema de reputación, resolución de disputas, gestión financiera...

A continuación, exponemos algunas de las ideas básicas que están detrás de Colony:

- *Tarea*: una tarea es **la unidad más pequeña** dentro de Colony. **No se puede dividir** y se puede evaluar como completa o incompleta en función de una serie de criterios. A cada tarea se le asigna un dominio y tiene que tener una descripción de cómo es y cómo se va a evaluar.

- *Reputación y Tokens*: la reputación es l**a forma que tiene la comunidad de saber los méritos conseguidos por una persona en concreto**. Se consigue con la realización de tareas financiadas con tokens internos de la empresa, y no se puede transferir. El cumplimiento de las tareas se puede pagar con tokens de pago (moneda con valor real fuera de la organización y que no aportan reputación al usuario que la completa) o con tokens internos de la organización, los cuales sí dan reputación. A diferencia de la reputación, **los tokens de pago sí pueden ser transferidos** entre cuentas y **sólo pueden ser ganados o perdidos mediante la resolución de tareas**, disputas etc. Para finalizar indicar que tanto los tokens de pago como la reputación **son necesarios para** crear y asignar tareas, votar y, en resumen, **poder**

participar dentro de la organización.

- *Dominios y Habilidades*: estas dos características permiten la división del trabajo sin tener que establecer una jerarquía de poderes. Los dominios son **subestructuras para dividir el trabajo de la empresa**, lo que sería lo mismo que los departamentos en las empresas tradicionales. Por otro lado, las habilidades **se encargan de categorizar el trabajo independientemente del dominio** al que pertenezcan.

- *Pots*: las organizaciones descentralizadas tienen los *pots*, que no son más que **montos de dinero asignados a un dominio**, para entenderlo, serían como los presupuestos asignados al departamento de marketing o al de desarrollo en una empresa convencional. Cómo mínimo, una colonia tendrá dos *pots* básicos: un ***pot* de recompensas** y otro para **capital de trabajo**. Para finalizar indicar que cuando se crea un nuevo dominio dentro de una colonia se le asigna un *pot* inicial y dicho *pot* solo puede ser gestionado por los administradores y fundadores de la colonia.

Por último, indicar que a día de hoy, Colony se encuentra en su versión llamada *glider* y tiene implementado las siguientes características: Tokens, *pots* de financiación, reputación, sistema de pagos, dominios de un nivel, habilidades, tareas, sistema de calificación de trabajo, roles y permisos a nivel de dominio.

3.4.3. Aragon

Aragon es el *framework* que utilizamos para construir nuestra Dapp. Es una iniciativa Española, nacida el 17 de mayo de 2017, y está entre las ICOs (Initial Coin Offering o en español OPV -Oferta Pública de Venta-) más importantes de la historia ya que recaudó 24 millones de dólares en tan solo 15 minutos. Aragon nos proporciona un conjunto de herramientas y componentes ya programados para poder crear una DAO de la mejor manera posible, además, los principales motivos por los que lo escogimos fueron que, en el momento de comenzar con el trabajo de fin de grado, era el **más avanzado de los tres**, ya que tenía

implementados una gran cantidad de componentes, como el sistema de votaciones o el administrador de tokens y el que **mejor documentación tenía** (incluía un tutorial de inicio y una *Dapp* de demostración). Para terminar, si quiere saber más sobre él, diríjase al capítulo 5 de este documento.

Capítulo 4

Metodología y tecnología

En este capítulo hablaremos de las metodologías que hemos utilizado, así como el plan de trabajo que hemos seguido, el tipo de organización de actividades para la realización del trabajo, como es el modelo de metodologías ágiles SCRUM y el enfoque de software libre que queríamos transmitir.

Por otro lado introduciremos todas aquellas tecnologías que hemos utilizado para la realización de nuestra *Dapp* y evitar que el lector de este documento tenga que investigar pos su cuenta sobre ellas.

4.1. Metodología

4.1.1. Plan de trabajo

Nuestra estrategia inicial consistió en **recabar información y aprender los nuevos lenguajes de programación** que íbamos a necesitar para la realización del proyecto. Toda la información que recuperábamos la íbamos anotando para que nos sirviera como apoyo en caso de dudas de algún tipo durante fases más avanzadas del proyecto, esta fase ocupó casi la mitad del tiempo del proyecto.

La siguiente fase del proyecto consistió en **reuniones semanales** junto con los tutores del proyecto en las que tratábamos de proponer unas **metas** para la semana siguiente y **resolver las dudas** que nos habían surgido durante la semana del trabajo. Este método de

trabajo lo hemos mantenido hasta el final ya que así conseguimos tener unas metas realistas que presentar a los tutores. Esta fase la dividimos en tres etapas de trabajo: la primera de ellas consistió en realizar una **especificación de requisitos**, la segunda se **implementaron los requisitos** ocupando la mayoría de la fase de trabajo y la última en la que **se realizó la memoria**.

4.1.2. Enfoque de Software Libre

Hemos elegido una licencia **AGPLv3**, ya que nuestro objetivo es que cualquiera que utilice nuestro código deba también liberar el código fuente para su utilización en aplicaciones web o en un servidor de red para SaaS o con el propósito de hospedaje en la nube.

El código fuente de nuestra DAO esta disponible en GitHub:

https://github.com/TFG-DAOs/developer-community

4.1.3. SCRUM

Hemos utilizado la metodología ágil SCRUM porque es lo que mejor se adapta a nuestro tipo de proyecto. Este método consiste en un conjunto de buenas prácticas para trabajar en equipo y obtener los mejores resultados.

La característica principal de este sistema de trabajo es que el desarrollo es incremental y va evolucionando en función de los objetivos que se van consiguiendo. Además, se divide en *Sprints* que son periodos de tiempo entre una y cuatro semanas,**en nuestro caso de una semana**, en los que se desarrollaba un incremento de software entregable y funcional.

Gracias a realizar el trabajo de esta manera, pudimos observar todas a las grandes posibilidades que hay dentro del mundo de las DAOs (economías colaborativas, sistema de transiciones, sistema de votaciones) y focalizarnos en el sistema de transiciones.

4.1.4. Contribuciones al proyecto

Adrián Guevara Carpizo

Los primeros meses del proyecto los destiné a la investigación y aprendizaje de las tecnologías que íbamos a utilizar. La primera en la que me formé fue Solidity ya que es la base del proyecto, realicé proyectos pequeños para poder familiarizarme con las estructuras y métodos que tiene este lenguaje. Después comencé a realiza aplicaciones integrando React que me sirvió para familiarizarme con la estructura que tiene una aplicación y como funciona por dentro.

Mientras aprendía los nuevos lenguajes y las nuevas tecnologías también estuve recopilando información acerca de las tres plataformas que teníamos en mente para poder realizar una DAO (DAOstack, Colony y Aragon). Una vez decidimos que la plataforma que íbamos a utilizar era Aragon comencé a investigar más a fondo sobre el framework y la manera de crear una aplicación desde cero. El mismo framework proporciona una aplicación de un contador funcional que sirve para realizar las primeras pruebas de una aplicación.

Conseguimos un documento del grupo Guerrilla en el que explicaban todos los requisitos necesarios para la realización de una DAO para su grupo y me dediqué a extraer todos para poder tener una idea más clara de lo que tendríamos que llevar a cabo. Comencé extrayendo todos los requisitos que hablaban de los usuarios y la gestión de miembros ya que era lo que teníamos en mente realizar. Después estuve abstrayendo esos requisitos para poder sacar un esqueleto que nos sirviera como primer modelo para la aplicación.

A continuación tuvimos reuniones en las que hicimos varios bocetos de la parte gráfica y entre todos conseguimos tener una idea clara de lo que queríamos y lo que necesitaríamos para poder realizar el modelo que queríamos realizar.

Después de varias reuniones conseguimos tener una idea inicial de Guerrilla Developers y

comencé a desarrollar las primeras funciones del Smart Contract. Concretamente desarrollé añadir perfil, eliminar perfil, añadir transición y eliminar transición. Cuando ya tenía las funciones desarrolladas comencé con la interfaz gráfica y la conexión entre ambas. La parte gráfica decidimos desarrollarla con los elementos que ya están predefinidos en Aragon en lugar de hacerlo con Bootstrap.

Tanto el Front-end como la parte de Redux está desarrollada con Javascript ES6 ya que tiene algunas diferencias que hace más fácil la interpretación de algunas estructuras y cuenta con bastantes mejoras respecto al anterior estándar. Para poder tener conocimiento acerca de este nuevo estándar estuve realizando algún curso ya que todo lo que yo había desarrollado estaba en ES5.

Las siguientes semanas las dediqué a pulir el código corrigiendo errores y realizando test del smart contract para comprobar que todas las funciones funcionan correctamente y están controladas todas las excepciones. A la vez también comencé a realizar junto con mis compañeros el primer índice de la memoria para poder empezar con ella.

Comencé con la memoria junto con mis compañeros y decidimos realizarla en Latex trabajando en Overleaf para poder hacerlo de forma colaborativa. Los apartados de la memoria que he realizado han sido:

- Capítulo 1: Completo.
- Capítulo 2: Completo.
- Capítulo 4: Plan de trabajo, Tecnologías, NodeJs y NPM, Javascript, Redux.
- Capítulo 5: Instalación, Problemas y Soluciones.
- Capítulo 6: Inicios
- Capítulo 7: Completo

- Capítulo 9: Completo (todos los integrantes)
- Capítulo 10: Completo (todos los integrantes)
- Bibliografia

Diego González Blanco

Durante los primeros meses del proyecto los invertí en realizar búsqueda de documentación sobre DAOs, Colony, DAOstack y Aragon para tener los conocimientos y enlaces de referencia a la hora de desarrollar el proyecto.

Una vez decidimos que utilizaríamos Aragon para desarrollar nuestra Dapp, estuve haciendo el tutorial que ofrece Aragon donde encontramos un contador (una aplicación muy sencilla donde poder entender como funcionan Solidity, Web3 y React entre sí y obtener así los conocimientos para comenzar una Dapp de cero con esta plantilla) tanto con contratos como la parte gráfica que podemos ejecutar como un nodo desde localhost.

Más adelante, comencé a desarrollar Guerrilla Developers pero me faltaban conocimientos de la relación entre componentes de Aragon y la información que transita entre ellos, por lo que recurrí al proyecto de P2P Models - Wiki, y como en el caso del contador es un proyecto en Aragon pero tiene mayor complejidad por lo que aprendí mucho sobre componentes y hooks, ya que, tenia estructuras de datos más complejas que el contador al igual que pasaba con sus componentes gráficos y tomándolo como referencia ya podía comenzar a desarrollar Guerrilla Developers.

Una vez tuvimos un primer boceto del contrato "ProfileManager" con las primeras funciones básicas que eran añadir perfil, añadir transición, añadir miembro y asignar un perfil al miembro. La manera más sencilla de comenzar con el proyecto era de forma progresiva por lo que para aplicar lo que había aprendido anteriormente con el contador y con la Wiki, lo puse en practica con añadir perfil, es decir, manejar el evento de la función, recogerlo

dentro del script.js que es donde se contiene el estado de la Dapp y así actualizarlo en base al evento. Después desarrollé la parte gráfica y la conexión entre el botón y el formulario asociado, con la función de añadir perfil.

La función con la que más problemas tuvimos fue la de añadir transición, en la que estuvimos trabajando Adrián Guevara y yo, ya que la información sobre mapping dentro de mapping era casi inexistente y nuestro objetivo era encontrar una manera de actualizar el estado con Redux, tal y como habíamos hecho anteriormente con funciones que utilizan estructura de datos más sencillas. Una vez entendimos como funcionaban estas estructuras y la información que teníamos que almacenar el desarrollo del resto de funciones fue mucho menos costoso.

Otro aspecto que estuve realizando fue corregir errores que nos comentaron en las reuniones, llevar a cabo el boceto de interfaz gráfica de manera que fuera lo más intuitivo posible y estuviera organizado con el correcto comportamiento de los componentes y las funciones del contrato.

Respecto a la memoria, la hemos ido haciendo de forma colaborativa entre los tres miembros del proyecto empleando OverLeaf y Latex. De este modo voy a indicar los apartados de la memoria de los que me he encargado que son los siguientes:

- Capítulo 2: Blockchain
- Capítulo 3: Enfoque de Software Libre, React.
- Capítulo 5: ¿Por qué utilizar Aragon?
- Capítulo 6 :Primer contacto: Contador, segundo contacto: P2PModels - Wiki.
- Capítulo 8: Añadir perfil, Eliminar perfil, Incrementar contador de contribuciones.
- Capítulo 9: Completo (todos los integrantes)

- Bibliografia

Manuel Antonio Fernández Alonso

Comencé investigando por aquello que salía en todos los lados cuando buscabas por desarrollar en Ethereum, que es solidity. Estuve bastante tiempo investigando sobre que tipo de lenguaje era y como se utilizaba. Ahí descubrí Remix que era un compilador online de este lenguaje. Más tarde encontré un libro sobre Ethereum, llamado "Mastering Ethereum", de Andreas M. Antonopoulos, Gavin Wood el cuál me ayudó a entender lo que erá Ethereum, como funcionaba, que eran los Smart Contracts y como empezar en esta red.

Más adelante, empecé a investigar sobre las DAOs, qué era y para qué servían. No estaba del todo claro cuando empezamos ya que era un concepto muy ambiguo de entender, pero según ha ido pasando el tiempo el concepto se ha ido afianzando y cogiendo un rumbo concreto. Nuestro TFG, en unos inicios, consistía en una investigación sobre los principales frameworks para crear una DAO, los cuales investigué. Fue un proceso arduo, ya que al ser una tecnología en su versión alpha, no existía demasiada información al respecto.

Después realicé un curso de Udemy en el que te explicaba más en detalle el lenguaje de programación de *smart contracts*, Solidity, y cómo relacionarlo con la parte *frontend* gracias a la librería de web3. En el me familiaricé también con *Mocha* que es la herramienta que te permite realizar test exhaustivos de los contratos.

Cuando tuvimos clara la idea del trabajo, me puse a investigar sobre una organización que sería la base de nuestro TFG llamada "Guerrilla Translator", de ella hice un un diagrama de estados con las posibles transiciones que existían entre los diferentes roles que la componían. Además teníamos que imaginar como sería la parte web, así que hice unos pequeños *mockups* de la misma.

Luego llegó el momento de investigar Aragon, cómo funcionaba y cómo eran las *dapps* que

nos ofrecía. Primero empecé con la *dapp* que te ofrecen ellos de demostración, un contador. Más adelante, nuestro tutor del trabajo, nos enseño una aplicación suya, una Wiki que te permitía añadir, editar páginas, borrarlas y también estuve trasteando con ella. El tema de trabajar con Aragon también me llevó a aprender una API de javascript llamada React que se utilizaba para toda la parte front.

A la hora de programar, tarde un tiempo a adaptarme al ritmo, y de la parte de programación del *smart contract*, en un primer momento solo realicé el "removeTransition". Mas adelante añadiría también la función de "addContributionToMember" y modificaria detalles, del resto de funciones como "removeProfile" o "addMember".

Con respecto a la parte del front, hubo un momento que se tuvo que hacer una remodelación del tipo de datos, una confusión que nos obligó a revertir el código y que tuve que hacer. También realicé el diseño final de la aplicación con aprobación de mis compañeros y nuestros tutores. La parte front de "AddMember", tanto la tabla, como el *badge* de los miembros, como el botón de assign y el de añadir contribución forman parte de mis aportaciones al proyecto. Por ultimo, realicé la parte de los tests para comprobar que ciertas funcionalidades del contrato se realizaban correctamente y me cercioné de que la función de añadir rol, realizada por los tres, funcionase correctamente.

Con respecto a la memoria, la hemos realizado conjuntamente y todos hemos aportado mucho trabajo y esfuerzo en ella. Aquí pondré los puntos en los que he participado:

- Capítulo 1: Aportación de un objetivo.
- Capítulo 3: Ethereum, Organizaciones Autónomas Descentralizas, Framework de implementación de DAOs.
- Capítulo 4: SCRUM, Metamaks, Web3, Solidity.
- Capítulo 5: ¿Qué es?

- Capítulo 6: Aportación al punto Inicios.
- Capítulo 8: Completo (excepto los puntos de Diego y en el planteamiento que fue común).
- Capítulo 9: Completo (todos los integrantes)
- Bibliografía

4.2. Tecnologías

Nuestro *framework* de desarrollo es Aragon que está en **continua actualización** y por ello decidimos desarrollar todo el código sobre la misma versión (0.6) a pesar de que pudieran aparecer nuevas versiones. Solidity, el lenguaje de programación utilizado en la parte *backend* del proyecto, también se actualiza con bastante frecuencia y decidimos utilizar una versión estable a partir de la 0.4.22

La aplicación está dividida en la parte *back-end* donde lo principal serían los *smarts contracts* y la parte *front-end* (interfaz gráfica) donde destacaríamos el uso de React, web3 y Redux. Los contratos inteligentes están desarrollados en Solidity, los cuales utilizan la blockchain de Ethereum. Nosotros utilizábamos una blockchain local de Ethereum mediante el uso de Metamask. La parte gráfica está desarrollada con React y para conectar ambas partes hemos utilizado la librería de web3 y Redux.

4.2.1. Metamask

Para poder gestionar las cuentas de la DAO, la red de Ethereum en la que estábamos trabajando y el balance de las cuentas, hemos utilizado **Metamask**[6]. Sobre todo, lo que nos permite Metamask, es la **ejecución de aplicaciones descentralizadas desde**

el navegador, sin la necesidad de tener un nodo Ethereum funcionando en tu ordenador.

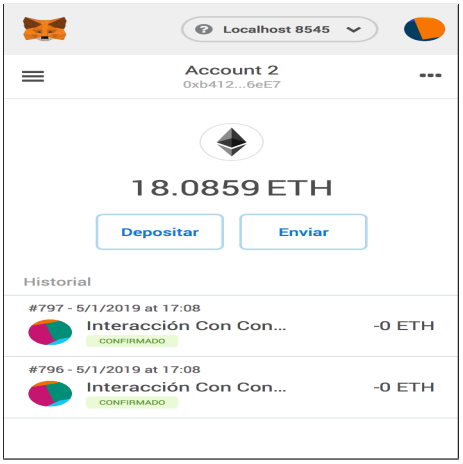


Figura 4.1: *Cartera de Metamask en Localhost, puerto 8545*

4.2.2. NodeJs y NPM

NodeJs[7] es un **intérprete de Javascript muy rápido y eficaz** que permite un gran número de conexiones abiertas y esperando simultáneamente. Además es un entorno de Javascript que utiliza un modelo asíncrono y dirigido por eventos que se organiza por paquetes y aquí es donde entra NPM.

NPM[8] es un **gestor de paquetes** para proyectos específicos, estos paquetes se instalan en local, en una carpeta concreta dentro del proyecto. Estos hacen de librerías que dotan a la aplicación que se está desarrollando de más posibilidades. Además de instalar los paquetes que necesitamos, nos permite tener un archivo, en nuestro proyecto, con las dependencias de forma que en otro terminal en el que tengamos el proyecto podamos instalar todos los paquetes que necesitamos para que funcione.

4.2.3. Solidity

Solidity[9] es un lenguaje de programación de alto nivel con cierto parecido sintáctico a Javascript. Este lenguaje permite **crear y desarrollar contratos inteligentes** que se ejecuten en la Máquina Virtual Ethereum. A pesar de ser un lenguaje muy parecido a JS tiene ciertos tipos de datos y características que lo hacen muy diferente al resto, como el tipo *address*, que sirve únicamente para direcciones de Ethereum o métodos como *require* que hace **comprobaciones de manera inmediata dentro de la Blockchain**.

Estas comprobaciones que se hacen sobre la Blockchain **no tienen coste** alguno pero si se quiere **actualizar la Blockchain tendra un coste** computacional llamado **Gas**. Este método de pagar por realizar transacciones a la blockchain permite que no se puedan hacer bucles infinitos en los Smart Contracts que puedan colapsar la blockchain

4.2.4. Javascript ES6

ECMAScript v6 (ES6)[10] es el estándar que hemos utilizado para la realización del proyecto, ya que tiene cambios bastante significativos y útiles respecto al anterior. El cambio más utilizado es el de la función *arrow* que hace que la declaración de funciones con cierta estructura sea más fácil y más visual.

```
// ES5
// Imaginemos una variable data que incluye un array de objectos
var data = [{...}, {...}, {...}, ...];
data.forEach(function(elem){
    // Tratamos el elemento
    console.log(elem)
});
```

Figura 4.2: *Ejemplo de array en ES5*

```
//ES6
var data = [{...}, {...}, {...}, ...];
data.forEach(elem => {
    console.log(elem);
});
```

Figura 4.3: *Ejemplo de array en ES6*

Todos estos cambios hacen que el código sea más legible y permite realizar ciertas cosas que antes suponían un mayor esfuerzo a la hora de programar.

4.2.5. React

React[11] es una biblioteca de JavaScript para construir interfaces de usuario de forma sencilla. Ofrece la posibilidad de manejar la lógica de la interfaz de usuario manejando eventos, observar como cambia el estado con el tiempo y como se preparan los datos para su visualización.

Para utilizar React no hace falta utilizar JSX, pero es muy útil porque después de compilarse, las expresiones JSX se convierten en llamadas a funciones JavaScript y se evalúan en objetos JavaScript.

Los elementos del DOM de los navegadores son diferentes de los elementos de React ya que estos últimos son objetos planos y su creación es de bajo costo. El DOM es actualizado por React DOM para igualar los elementos de React. Hay que diferenciar entre elementos de React y componentes de React, los elementos son los que forman los componentes.

Se pueden definir componentes como si se tratasen de una función de JavaScript, donde podemos pasar por argumento de la función las variables con los datos que se quieran mostrar. También se puede utilizar una clase de ES6 heredando de React.Component.

En esta tecnología se puede agregar un estado local a una clase, para ello hay que añadir un constructor de clase, como veremos en el ejemplo siguiente; se ha declarado el estado que contiene una propiedad date que contiene una nueva fecha:

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

Figura 4.4: *Ejemplo de Clase en ES6*

A su vez existen los "métodos de ciclo de vida" que son funciones creadas dentro de la clase con el objetivo de poder modificar los valores del estado local actualizando en este caso el campo "date" mencionado anteriormente. Un componente padre puede elegir pasar su estado como propiedades a sus componentes hijos, pero no es de importancia, para la clase, saber de donde vienen los datos, porque no tienen porque saber si otros componentes tienen estado o no ya que este es local y el flujo es hacia abajo, unidireccional.

La última actualización de React (v16.8.0) contiene los "Hooks", permiten utilizar el estado y otras características de React sin escribir una clase. Estos proporcionan una API más directa a lo que hemos mencionado anteriormente llamado "props" o "propiedades", estado, ciclo de vida, contexto y referencias.

Los Hooks permiten reutilizar la logia de estado sin cambiar la jerarquía del componente, esto facilita compartir variables y propiedades entre componentes. La siguiente imagen es un ejemplo de utilización de hooks:

```jsx
import React, { useState } from 'react';

function Example() {
  // Declara una nueva variable de estado, que llamaremos "count".
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Figura 4.5: *Ejemplo de hooks en ES6*

4.2.6. Redux

Redux [12] es una herramienta para la **gestión de estados de apps** en Javascript. Normalmente esta tecnología se relaciona directamente con React aunque no tiene porque ser siempre así. Un **estado** dentro de una aplicación puede ser el **conjunto de valores que queremos almacenar para poder acceder a ellos en un momento determinado**.

Con Redux hacemos que este estado se **modifique cuando y como queramos** teniendo así el máximo control, mientras que si no contamos con Redux tenemos los aspectos de mutabilidad y asincronismo propios de un *frontend* moderno.

```
case "AddTransition":
  {
    const finalProfile = toUtf8(event.returnValues.finalProfile);
    const initialProfile = toUtf8(event.returnValues.initialProfile);
    const timeCondition = event.returnValues.timeCondition;
    const contributionCondition = event.returnValues.contributionCondition;
    const initToFinalProfileExists = true;
    const conditions = { initToFinalProfileExists, timeCondition, contributionCondition }
    newState = {
      ...state,
      transitions: {
        ...state.transitions,
        [initialProfile]: {
          ...state.transitions[initialProfile],
          [finalProfile]: conditions,
        },
      },
    }
  }
```

Figura 4.6: *Ejemplo estado de la función "AddTransition" en Redux*

El estado que tenemos de una app en un momento determinado sólo puede alterarse emitiendo una acción que describa la nueva estructura del objeto, esta modificación se hace en unas funciones determinadas llamadas *reducers*. De este modo el flujo de una acción con Redux sería de la siguiente forma:

1. Un componente **recibe** un evento y **emite una acción**
2. Esta acción **modifica** el **estado**
3. La acción es **comunicada** junto con el estado actual **a los reducers**
4. Los reducers devuelven **el nuevo estado** a los componentes.

4.2.7. Web3

Es la principal librería de JS para interactuar con Ethereum en la parte del navegador. Nos aporta una manera de construir nuestro sitio web que es capaz de comunicarse con nuestra red de Ethereum. Es capaz de comunicarse con la red de Ethereum gracias a un **protocolo de comunicación llamado JSON-RPC**. Esto no es más que un protocolo de llamada, codificado en JSON. Se utiliza este protocolo para hacer **requests de un nodo en particular de la red de Ethereum y conseguir la información** que se necesita.

En nuestro proyecto hemos utilizado web3[13] para crear tests que comprueben el buen funcionamiento de nuestros *smarts contracts* y para funcionalidades como convertir strings a hexadecimal y viceversa. También lo hemos utilizado para *hashear* cadenas con SHA-3 ya que era necesario por el tipo de estructura de datos que utilizábamos en los contratos.

Capítulo 5

Aragon

En este capítulo se realizará una descripción de qué es Aragon y cuáles son las partes que lo componen. Se explicará mas en detalle "AragonOS", el sistema de actualizaciones, de control de permisos y cómo se comunican los componentes entre ellos. Además también se aclarará cual es la utilizad de "AragonAPI" y para que sirve "AragonUI". Después se darán los diferentes motivos por los que se ha elegido este *framework* para la realización del proyecto. Por último se explicará cómo se instala y los diferentes problemas que surgieron a la hora de hacerlo.

5.1. ¿Qué es?

Aragon[14] es un proyecto creado con la idea de **fomentar la gobernanza descentralizad**a, para que las personas que forman comunidades, **puedan organizarse sin intermediarios que la centralicen**. También trabaja con Blockchain por lo que utiliza contratos inteligentes que omiten gran cantidad de tramites, lo que añade simplicidad a su función.

Se originó en noviembre de 2016 donde se buscaba desarrollar una plataforma y una serie de herramientas que **faciliten** la creación de DAOs **enteramente democráticas y transparentes** donde toda información pertinente está a disposición de cada integrante de la organización.

Aragon pretende crear **una capa de usabilidad sobre Ethereum** de la misma forma que esta lo hizo al crear una plataforma que abstrae a los desarrolladores de los aspectos criptoeconómicos y de seguridad de los bloques del blockchain.

Aunque el objetivo principal del proyecto se centra en las DAOs, también se han **desarrollado y modularizado otros componentes** que pueden llegar a tener otras aplicaciones **como el desarrollo de Dapps o protocolos fundamentados en la criptografía** utilizando el cliente de Aragon. Los componentes son los siguientes:

AragonOS

Es un *framework* orientado al desarrollo de contratos inteligentes que busca a**bstraer al desarrollador del como la aplicación debe gestionar los recursos** de la organización y como **las entidades pueden acceder a estos**.

La filosofía de Aragon gira en torno al desarrollo de **aplicaciones altamente modularizadas y desacopladas** que poseen funcionalidades **bien definidas** y son capaces de **comunicarse entre sí**. Por este motivo, el *framework* ofrece las siguientes funcionalidades:

- *Actualización*:

  El *framework* ofrece métodos sencillos mediante los cuales podemos actualizar nuestra aplicación añadiendo nuevas funcionalidades o características, o reparando bugs que se encontraban en el código.

- *Control de Permisos*:

  Facilita al desarrollador la **asignación de permisos a entidades y otras aplicaciones**, gestionando quién puede acceder a las distintas funcionalidades. También se **abstrae de la lógica de autenticación**.

  Toda la actividad de gestión y control de permisos recae en la Lista de Control de Acceso (ACL), un componente del Kernel que provee "AragonOS" donde se encuentran **definidas todas las reglas de acceso**. Un ejemplo de un ACL sería el siguiente:

Entidad	App	Rol	Gestor
AppEjemplo	AppEjemplo2	ROL_EJEMPLO	AppEjemplo2

Podemos leer la lista de la siguiente manera: Una entidad AppEjemplo posee los permisos para ejecutar funciones de AppEjemplo2 bajo el rol ROL_EJEMPLO. Los permisos son aprobados o revocados por AppEjemplo2 Ahora, si se quiere proteger una función creada para algún contrato podemos agregar el modificador auth()

- *Comunicación entre componentes*:

"aragonOS" también facilita la comunicación entre las aplicaciones que desarrollemos al enviar peticiones de acción que pueden o no ser aceptadas de acuerdo a los permisos que la aplicación emisora posea.

Las aplicaciones desarrolladas con Aragon **no utilizan constructores para crear los contratos**. Debido a esto, se debe definir una **función "inicializadora"** para encargarse de todas las operaciones necesarias durante la creación del contrato.

AragonAPI

Esta API **nos permite interactuar con "aragonOS" y los estados de los contratos inteligentes** a través del objeto de la clase AragonApp, el cual se comunica con el wrapper de la aplicación por medio de un proveedor de mensajes que podemos configurar en función de si queremos comunicarnos con la aplicación desde scripts en segundo plano o desde las vistas de la interfaz de usuario.

AragonApp también provee una interfaz sencilla para poder comunicarnos con los contratos inteligentes anteriormente mencionados, llamando a los métodos de este del mismo modo como si fuera una función de JavaScript

La API también nos permite desarrollar scripts para ser ejecutados en segundo plano con el objetivo de **preparar, inicializar o mantener actualizado** el estado de tu aplicación.

AragonUI

Aragon también ofrece una gran cantidad de componentes para **crear las interfaces de usuario de las aplicaciones** desarrolladas.

En resumen y para finalizar, Aragon nos ha dado las facilidades para iniciar la programación de una DAO, ya que nos ofrece una serie de herramientas, como un contador, una aplicación para realizar votaciones y otra para tener un sistema de reparto de tokens las cuales nos han facilitado mucho el desarrollo.

5.2. ¿Por qué utilizar Aragon?

El conjunto de herramientas de Aragon sirve para desarrollar software con el propósito de conseguir **una DAO de uso humano**. Aragon ofrece una infraestructura que permite crear **aplicaciones resistentes a la censura, descentralizadas y actualizables**.

Por motivos de cantidad de información, **madurez** del framework y facilidad para comprender las tecnologías que utilizar para desarrollar una DAO, Aragon es la mejor opción frente a DAOStack y Colony, ya que cuando empezamos el trabajo de fin de grado todavía estaban por madurar en su desarrollo, en el caso de DAOstack estaba en sus primeras versiones y era difícil encontrar documentación relacionada, en el caso de Colony pasaba lo mismo. Pero Aragon nos ofrecía de manera ordenada bastante información de uso y tutoriales con los que poder empezar a comprender su funcionamiento.

5.3. Instalación

Para instalar Aragon lo primero que tenemos que hacer es instalar las versiones estables más actualizadas de Node.js y NPM sobre un sistema Linux. A continuación instalaremos AragonCli a través de NPM. Esto nos permitira tener una carpeta *./aragon* que contiene la informcacion de las cuentas de prueba que utilizaremos en nuestra blockchain local.

5.3.1. Problemas y soluciones

Quedarse sin fondos de Ether

En alguna ocasión las cuentas creadas por Aragon pueden llegar a quedarse sin fondos para poder hacer transacciones, en este caso debemos eliminar la carpeta */.aragon*.

IPFS

A la hora de ejecutar la aplicación nos puede surgir un problema con Ipfs, e indicarnos que no se encuentra, para solucionarlo debemos acceder a la pagina de ipfs e instalar la versión correspondiente para tu sistema operativo.

Falta de permisos

Cuando ejecutamos la aplicación nos puede dar un fallo de que no tenemos permisos suficientes para crear carpetas que son necesarias. En este caso, debemos asegurarnos de que la instalación que hemos hecho de Aragon ha sido sin permisos root y que el proyecto está en una carpeta con permisos de escritura.

Problemas con Ganache

Si obtenemos un problema durante la ejecución que nos da un mensaje acerca de Ganache debemos borrar la carpeta. */.aragon*. Ganache se encarga de crear una blockchain local de pruebas.

Capítulo 6

Primeros desarrollos

Antes de desembocar en la idea de **Guerrilla Developer** tuvimos varios proyectos en mente que no llegamos a desarrollar por diferentes motivos en este apartado se explica porque desechamos cada uno de ellos y cual fue el proceso a través del cual se desarrolló Guerrilla Developer.

6.1. Inicios

La primera de las ideas que tuvimos era hacer una plataforma que permitiera la **compraventa de artículos** de todo tipo, desde un lápiz a una casa. Esta idea nos llamó la atención porque para artículos pequeños ya existen varias plataformas pero tienen el problema de la **confianza comprador-vendedor**. Esto generaba un **problema** más complicado de resolver ya que nos obligaba a incluir un **perfil de intermediario** para validar los acuerdos de artículos, de un gran importe, como por ejemplo casas, coches, fincas... Ahí entraría el uso de ***smart contract*** porque nos permitían **suprimir a los intermediarios** necesarios, ya fuesen asesores, gestores, notarios... Pero esta idea tenía un único y **gran problema:** para realizar estos contratos deberíamos contar con el apoyo de alguien que conociera **la legalidad vigente** en el tema de compraventa y esto suponía la dependencia de una persona externa y su disponibilidad para ayudarnos así que descartamos esta idea.

Otro concepto interesante para desarrollar era hacer una plataforma de ***crowfunding***,

este concepto de aplicación ya existe, pero no implementado con contratos inteligentes, así que nos pareció la mejor idea para desarrollar. El problema surgió cuando nos dimos cuenta que en el *framework* que íbamos a utilizar había aplicaciones de ejemplo en las que se encontraba una de *crowfunding*. Intentamos buscar una nueva funcionalidad a la aplicación que nos permitiera darle una vuelta de tuerca pero finalmente acabamos desechando esta idea porque no encontrábamos esta funcionalidad.

Después de desestimar las ideas anteriores vimos que Aragon tenía un **sistema de roles** que podría tener una importancia dentro de una DAO con un sistema de economía colaborativa. Tras establecer que queríamos desarrollar una aplicación en las que los **roles fueran la clave de su funcionalidad** tuvimos acceso a un documento de Guerrilla Media Collective.

En este documento, que trataba de una organización de traductores, estaban **definidos los roles** de los usuarios de la organización así como los **requisitos** necesarios para cambiar de un rol a otro.

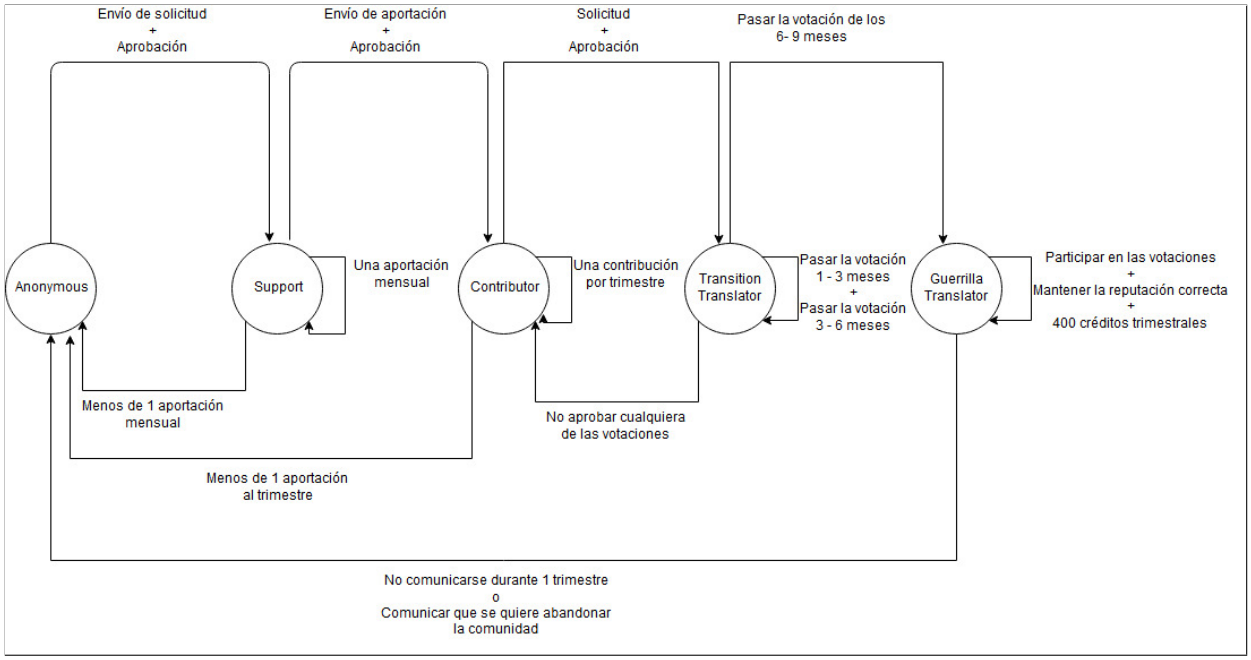


Figura 6.1: *Diagrama de estados de referencia*

Con esto decidimos hacer un **diagrama de estados** indicando los roles que existían en la organización e indicando los hitos que había que superar para hacer la transición de un rol a otro o para quedarse en el mismo rol. El mismo documento también incluía el **reparto de dinero** en la organización según el rol que tuvieses en la misma y la reputación, pero esta parte la descartamos porque daba para hacer otro trabajo de fin de grado.

6.2. Primer contacto: Contador

El primer modelo de DAO que podemos encontrar en el tutorial que ofrece Aragon para comenzar a desarrollar DAOs sacamos como obtener el proyecto en su primera versión. A partir del código pudimos identificar que archivos iban a ser clave para desarrollar nuestra ***dapp***.

Tras una reunión, obtuvimos un proyecto del contador que estaba en la versión de Aragon en la que ibamos a desarrollar el proyecto y obtenidos los conocimientos de que había que hacer, nos pusimos a manipular el código que podemos encontrar en: Boilerplate. A partir de aquí, se comenzó a desarrollar nuestro **smart contract** acorde con nuestra *dapp*.

Figura 6.2: *Apariencia de aragon-react-boilerplate una vez ejecutado.*

6.3. Segundo contacto:P2PModels - Wiki

La segunda referencia obtenida tras una reunión fue el proyecto de **P2PModels - Wiki**, que podemos encontrar en: Wiki. Este segundo contacto fue más difícil de asimilar debido a la complejidad del proyecto, aquí encontramos más ejemplos de la utilización de los **Hooks**, propiedad de React mencionada anteriormente, y mucha información sobre como conectar y comunicar la información entre los componentes.

Por ejemplo, en la siguiente imagen el recuadro rojo es una página de la wiki y el recuadro azul es un índice donde navegar por el resto de páginas, ambos son componentes diferentes pero están conectados por un padre que los contiene y maneja la **información entre componentes**.

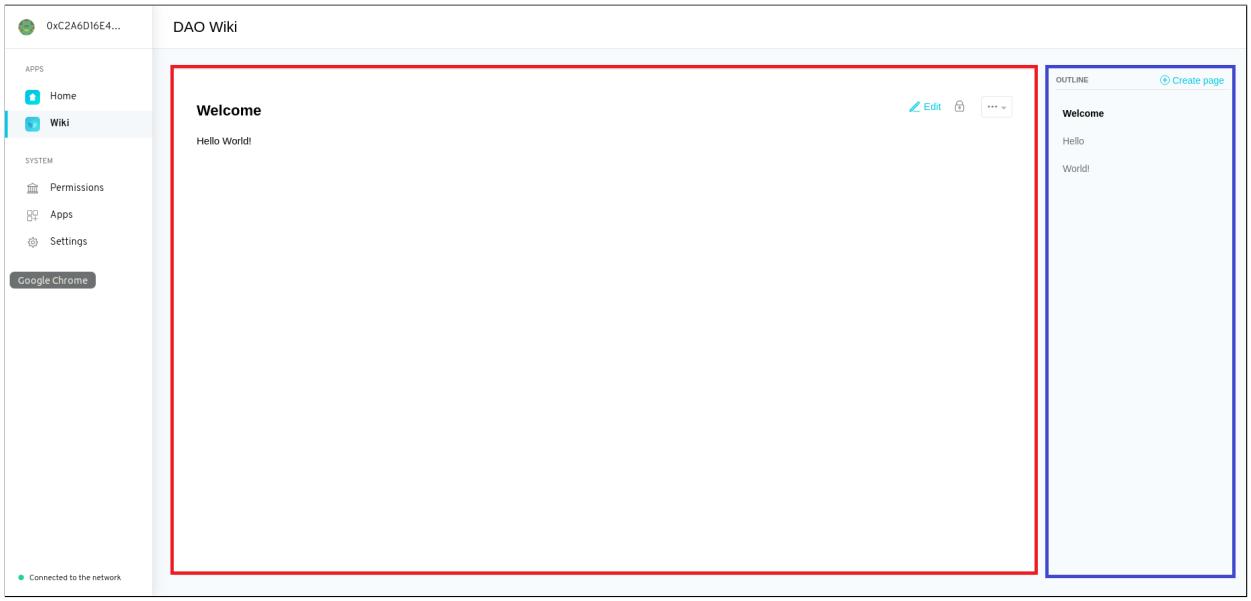


Figura 6.3: *P2P Models - Wiki: conexión entre pagina e índice de páginas.*

Capítulo 7

Guerrilla Developer

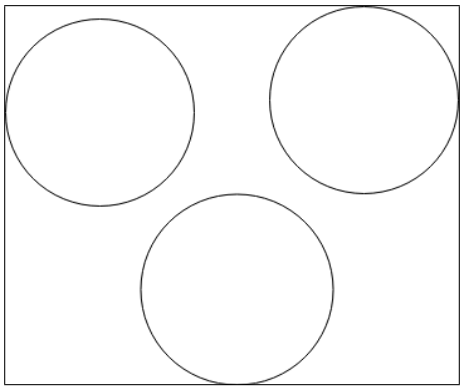
Una vez analizado el documento y extraído la información más importante decidimos que lo mejor sería intentar hacer una **aplicación que fuese genérica** para cualquier tipo de organización, ya sea una de traductores, una comunidad de vecinos, o una compañía de desarrollo de software (ahí nació "Guerrilla Developer", el concepto que utilizamos para desarrollar nuestra dapp), así que decidimos abstraer la información que habíamos recolectado y desarrollar una ***dapp* genérica de transición de perfiles**. Utilizamos ahora la palabra **perfiles**, porque dentro de Aragon existen los roles, y no tienen nada que ver con el concepto que nosotros queríamos, así que decidimos llamarlo perfiles, siendo estos los estados del diagrama de estados, por ejemplo, en la organización de traductores, los perfiles serían "support", "contributor", "transition translator" y "guerrilla translator".Por último, 'para llegar a estos perfiles hay que cumplir una serie de **objetivos** como **permanecer un cierto tiempo en un perfil anterior** y **traducir un cierto número de textos**. Esos objetivos serían las **condiciones** para pasar de un perfil a otro.

7.1. Tipos de organización

Analizando el documento de "Guerrilla Translator" empezamos a pensar en los diferentes tipos de organizaciones que pueden existir y como crear un esqueleto que pudiera adaptarse a todos. Diferenciamos entre tres tipos de organización que se pueden resumir en roles independientes, roles concéntricos y roles mixtos.

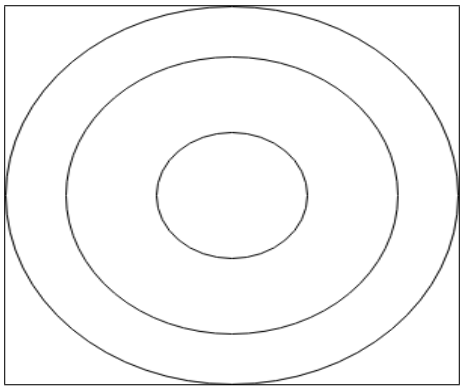
Roles independientes

Este tipo de organización cuenta con **roles totalmente diferenciados** entre sí, es decir, cuando un miembro de la organización tiene un rol determinado tiene unas **funciones concretas** y otro miembro con otro rol tiene otras funciones completamente diferentes. En este tipo de organizaciones no existe un **ascenso** vertical sino una **especialización**.



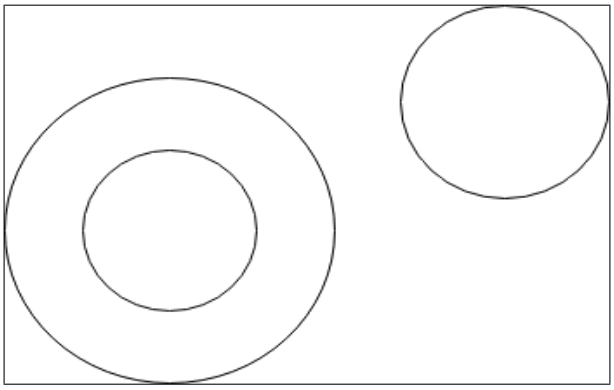
Roles concéntricos

En este tipo de organizaciones un rol tiene unas funcionalidades y cuando cambias de rol es para ascender o descender, es decir, cuando **desciendes** de rol **pierdes** algunas de las **funciones** que tenías y cuando **asciendes ganas funciones**. En este caso el **ascenso es vertical** ya que cuando llegas al último rol tienes las funciones de todos los demás roles juntos.



Roles mixtos

Este tipo de organizaciones **mezclan los dos tipos** que acabamos de explicar, existen roles en los que el **ascenso es vertical** y **roles** que están **totalmente separados** de estos con funciones específicas.



7.2. Especificación de requisitos

7.2.1. Requisitos estructurales

Los datos se tratan para guardarlos en la blockchain y también en la *store* del frontend por tanto la estructura puede ser diferente aunque el dato sea el mismo.

Miembros

Tanto en el **contrato** como en el **estado** de la app se trata de un **mapa** con la **dirección** de Ethereum como **clave** y una **estructura** como **valor**

```
struct Member {
    bytes32 profile;
    uint256 creationDate;
    bool exists;

    uint256 contributions;
}
```

Figura 7.1: *Estructura de Member*

```
mapping(address => Member) members;
```

Figura 7.2: *Mapping de direcciones de los miembros*

Perfiles

En el **contrato** se trata de un **mapa** con el **nombre del perfil** como **clave** y un **booleano** como **valor** mientras que, para el **estado** de la app se tratara como un **array** con los **nombres** de los perfiles existentes.

```
mapping(bytes32 => bool) public profiles;
```

Figura 7.3: *Mapping de perfil-bool de los perfiles*

Transiciones

En el **contrato** se trata de un **mapa** que utiliza como **clave** un **hash** único y como **valor** las **condiciones** de esa transición. En el **estado** de la app se guardará como un **mapa de un mapa** en el que la **clave** del primero es el **perfil de inicio** de la transición y la **clave** es otro **mapa** cuya **clave** es el **perfil destino** y **valor** las **condiciones** de la transición.

```
mapping(bytes32 => Conditions) transitionRegister;
```

Figura 7.4: *Mapping de la clave hash (inicio->destino) de la transición con las condiciones*

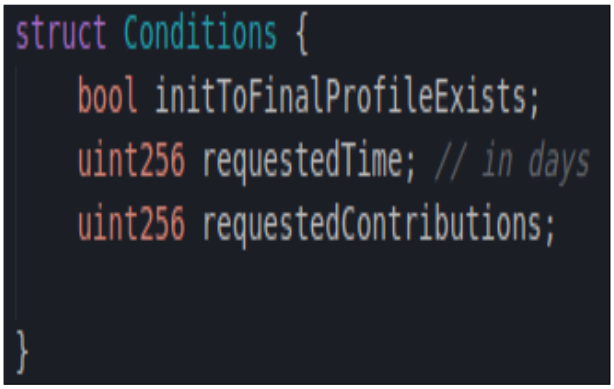


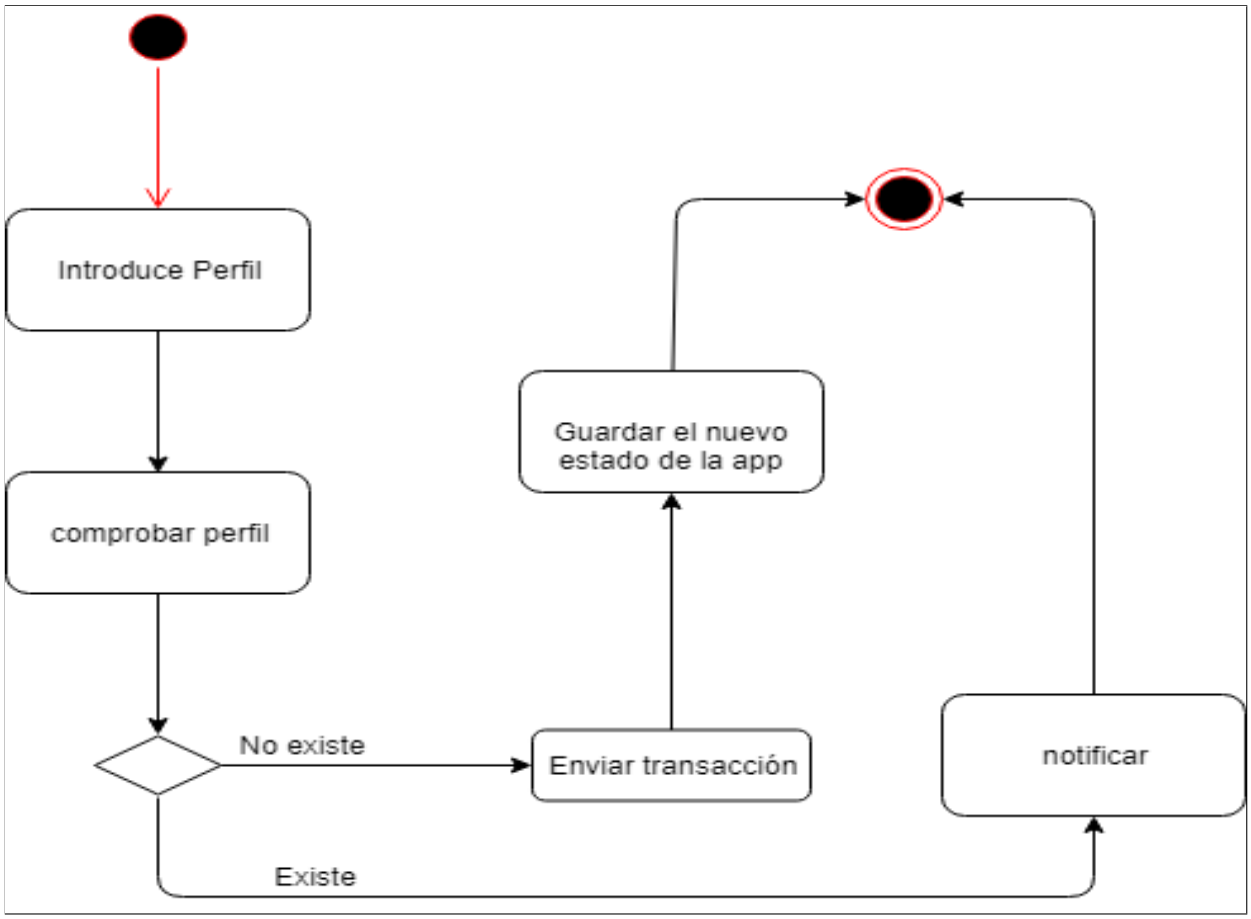
Figura 7.5: *Estructura de Conditions*

7.2.2. Requisitos funcionales

Tras analizar las estructuras necesarias para el sistema de gestión de usuarios procedimos al análisis de los requisitos funcionales necesarios para la aplicación, los cuales intentamos que fuesen los más básicos posibles para que se pudieran adaptar a cualquier organización de manera sencilla.

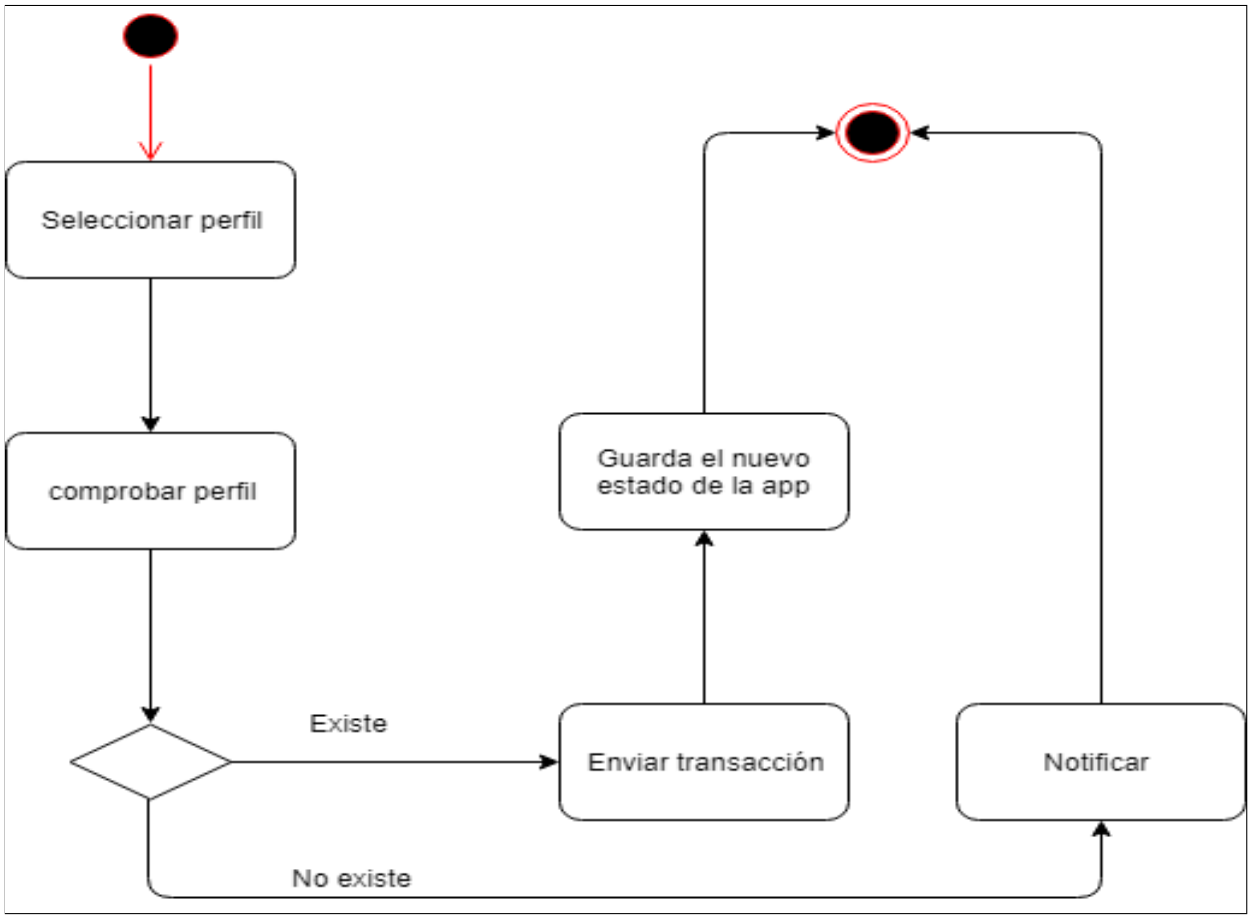
RF-1 Añadir Perfil

- Descripción: añade un nuevo perfil
- Entrada: nuevo perfil
- Salida:
- Necesita: fondos suficientes, conexión a la blockchain
- Precondición: no debe existir el perfil
- Poscondición: perfil añadido
- Excepciones: Si ya existe ese perfil se le comunica al usuario


Introduce Perfil
comprobar perfil
No existe
Existe
Enviar transacción
Guardar el nuevo estado de la app
notificar

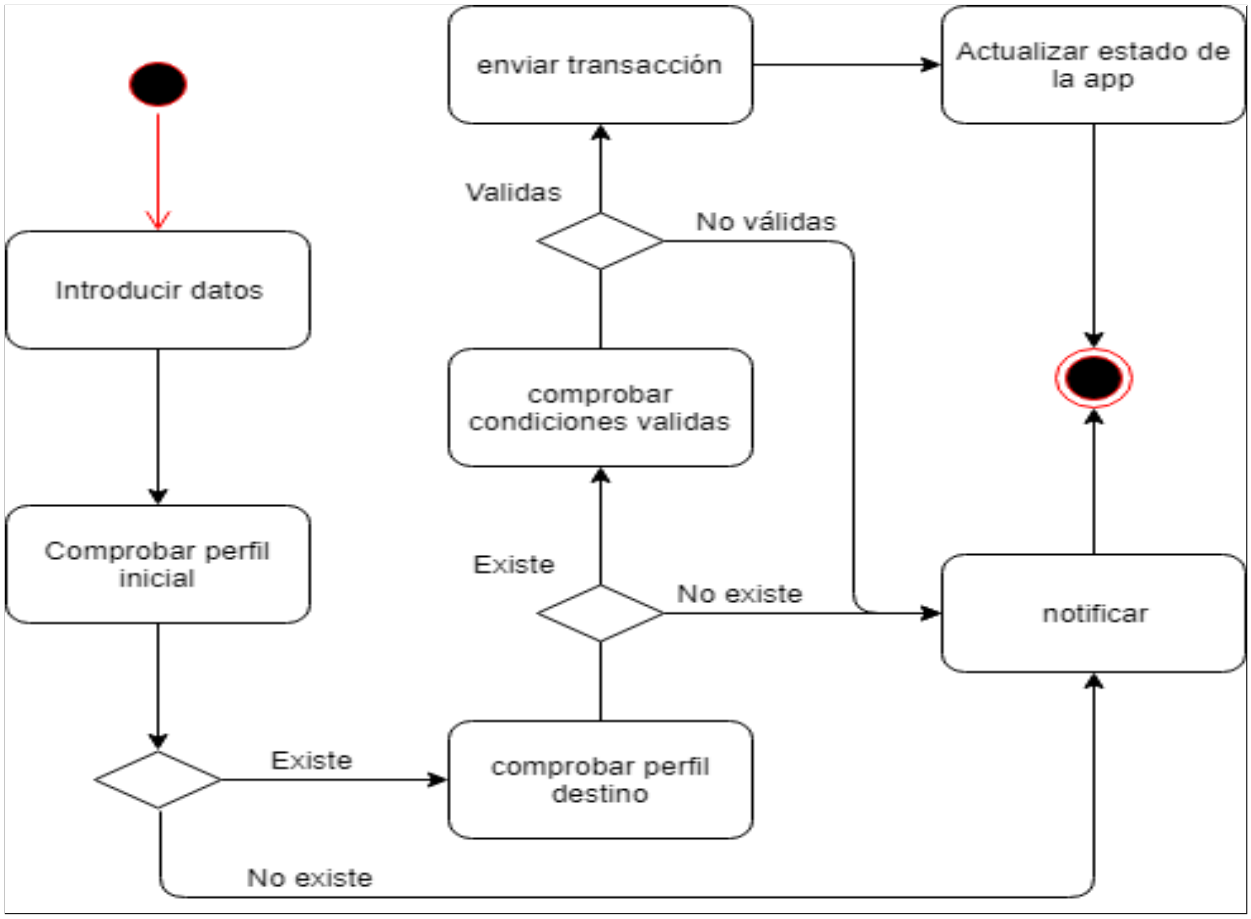
RF-2 Eliminar Perfil

- Descripción: elimina un perfil de la lista
- Entrada: perfil a eliminar
- Salida:
- Necesita: fondos suficientes, conexión a la blockchain
- Precondición: el perfil debe existir
- Poscondición: perfil eliminado
- Excepciones: si no existe ese perfil se le notifica al usuario



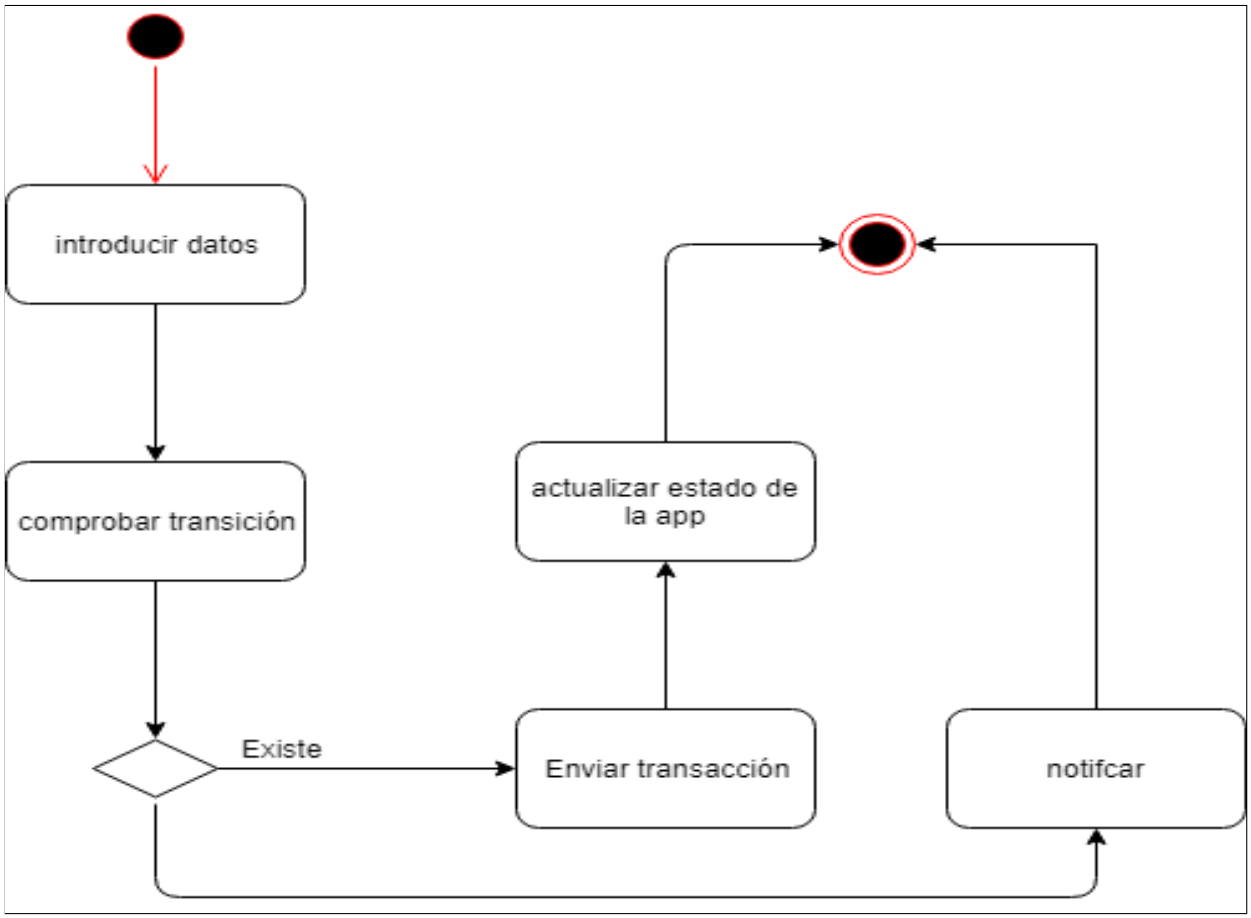
RF-3 Añadir Transición

- Descripción: añade una transición entre dos perfiles con unas condiciones determinadas
- Entrada: perfil inicial, perfil destino, condición de tiempo, condición de contribuciones
- Salida:
- Necesita: fondos suficientes, conexión a la blockchain
- Precondición: los dos perfiles deben existir
- Poscondición: transición añadida
- Excepciones:



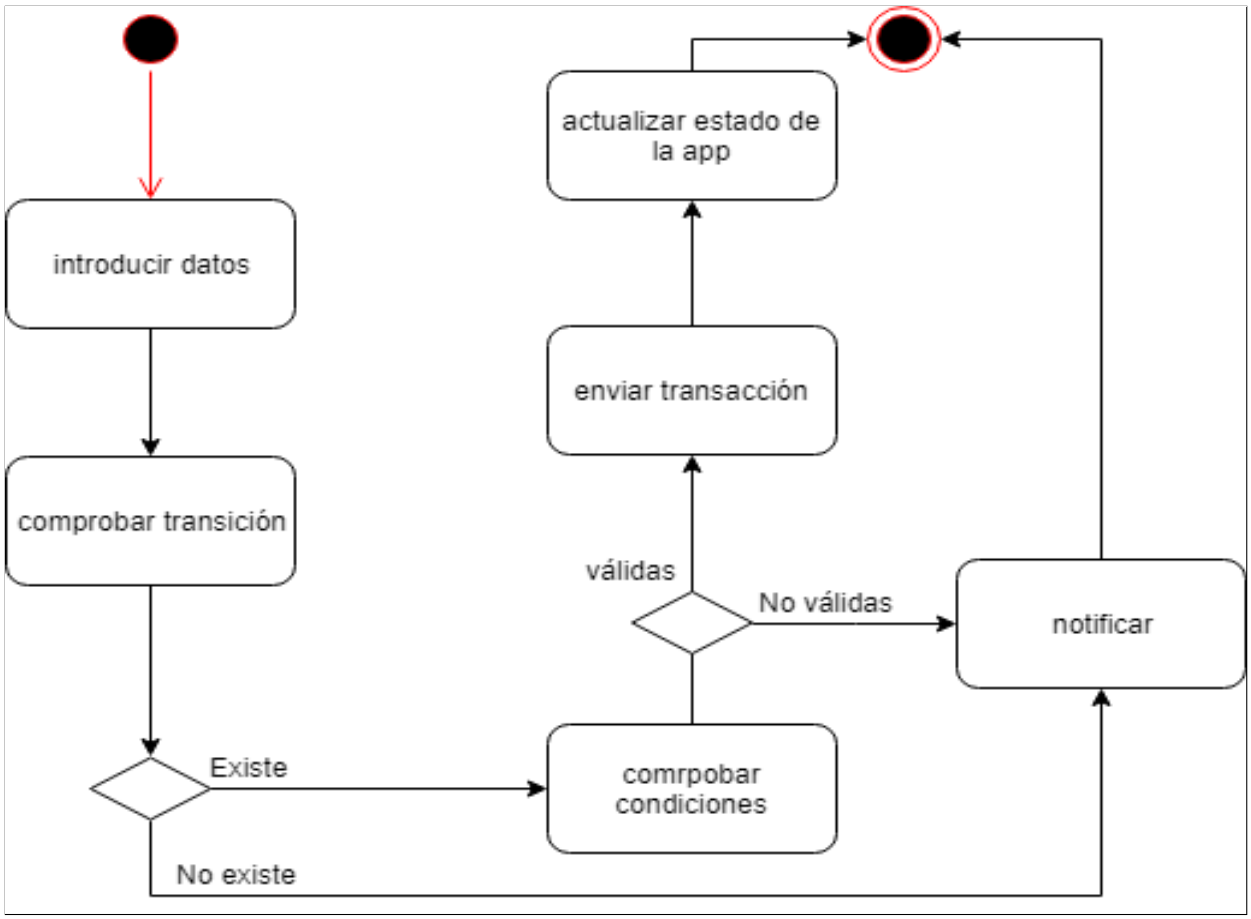
RF-4 Eliminar Transición

- Descripción: elimina una transición dados dos perfiles
- Entrada: perfil inicial, perfil destino
- Salida:
- Necesita: fondos suficientes, conexión a la blockchain
- Precondición: debe existir una transición entre los dos estados
- Poscondición: transición eliminada
- Excepciones:



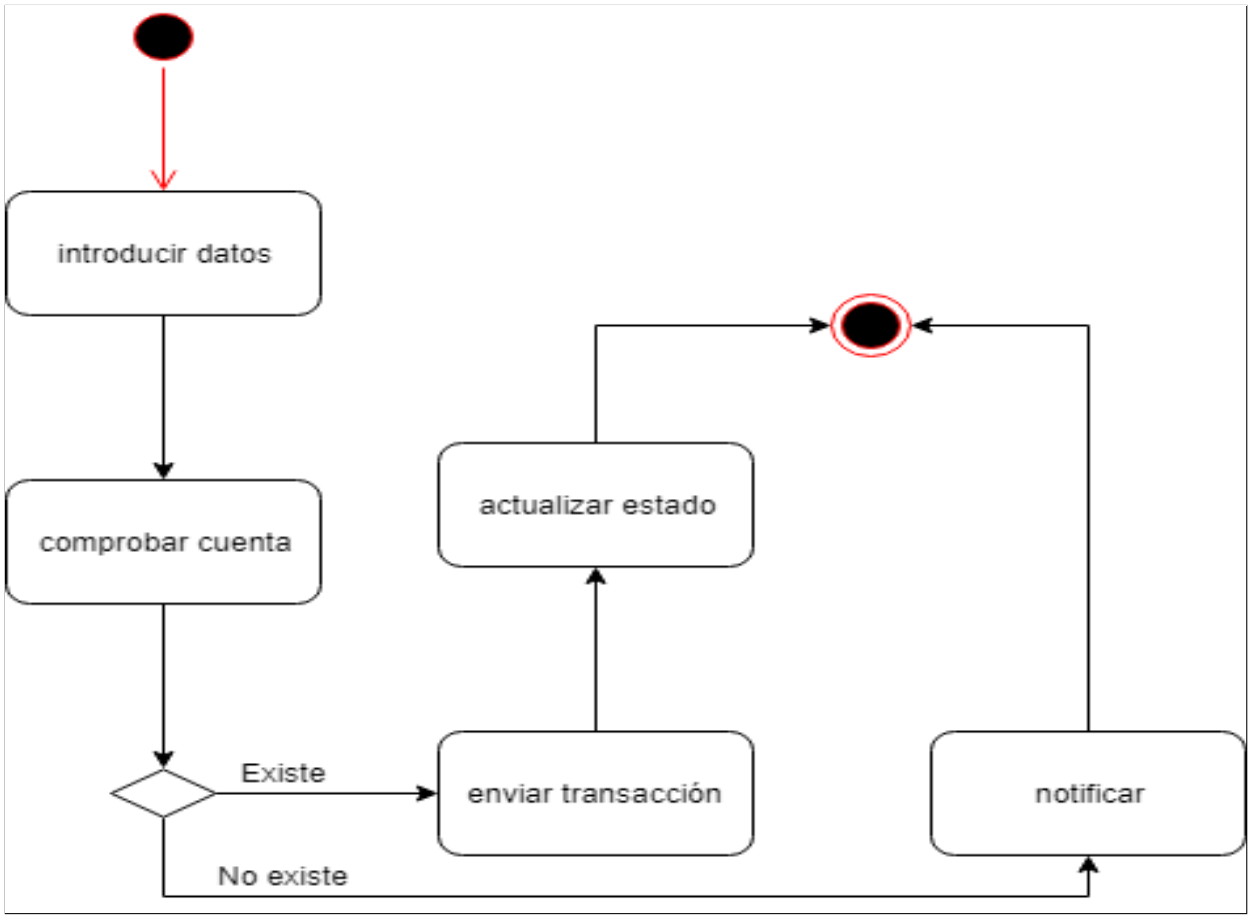
RF-5 Modificar Transición

- Descripción: modifica las condiciones que hay en una transición
- Entrada: perfil inicial, perfil destino, nueva condición de tiempo, nueva condición de contribuciones
- Salida:
- Necesita: fondos suficientes, conexión a la blockchain
- Precondición: debe existir una transición entre los dos estados
- Poscondición: muestra las condiciones modificadas en la tabla de transiciones
- Excepciones:



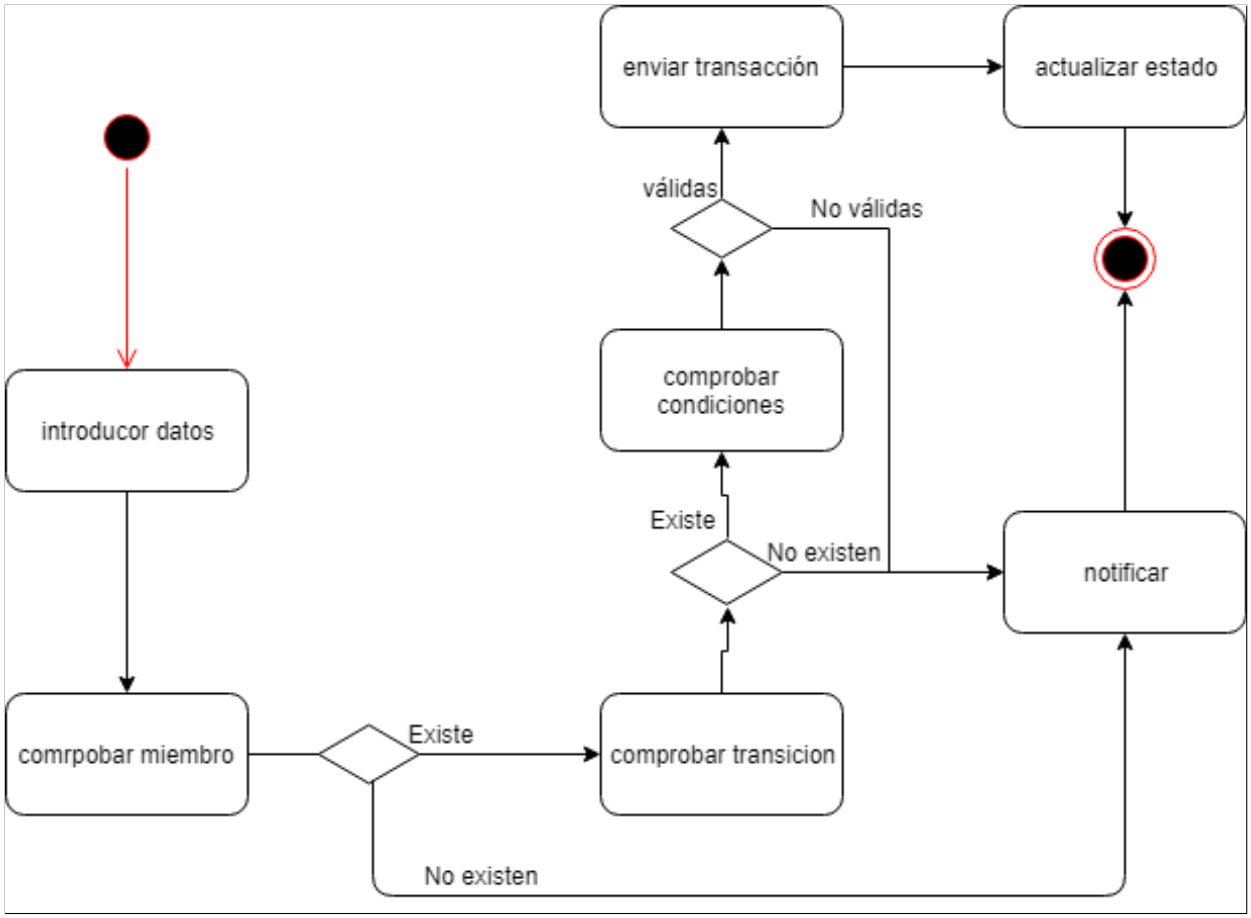
RF-6 Añadir Miembro

- Descripción: añade un miembro nuevo a la organización
- Entrada: cuenta de Ethereum
- Salida:
- Necesita: fondos suficientes, conexión a la blockchain
- Precondición: no debe estar ya en la organizacion y debe existir el perfil por defecto
- Poscondición: muestra el miembro en la tabla del perfil por defecto
- Excepciones:si ya existe en la organización se le notifica al usuario



RF-7 Asignar Perfil a Miembro

- Descripción: asigna un nuevo perfil a un miembro
- Entrada: miembro y nuevo perfil
- Salida:
- Necesita: fondos suficientes, conexión a la blockchain
- Precondición: debe existir el miembro, debe existir una transición al nuevo estado y el miembro debe haber cumplido las condiciones
- Poscondición: muestra al miembro en la tabla del perfil
- Excepciones:



Capítulo 8

Aplicación de Transición de Perfiles

En este capítulo se explica el funcionamiento en detalle de todas las funciones que ofrece la Dapp que se ha implementado para el *framework* de Aragon. Además se indican las funcionalidades en las que se pueda producir una excepción, qué excepción se produce y por último, un punto final con la explicación de ciertos tests que realizamos para la comprobación del correcto funcionamiento del *smart contract*.

8.1. Objetivo

El objetivo de esta aplicación es **automatizar** el trabajo de gestión de perfiles dentro de una **organización descentralizada**. Gracias a nuestra aplicación se podrán establecer unas reglas básicas para poder transitar de un perfil a otro y ella misma será capaz de reconocer cuando se han cumplido **los requisitos de las transiciones** posibles y lanzar un aviso. Por ejemplo, si un miembro se encuentra en el perfil *Anonymous* y para transitar al perfil *Support* necesitas 1365 contribuciones y 2 meses de tiempo en el perfil *Anonymous*, nuestra Dapp se encarga tanto de establecer los requisitos de esa transición, como de comprobar que luego los usuarios con el perfil *Anonymous* los cumplan para poder transitar.

8.2. Implementación

A continuación, se explicará la interfaz gráfica de la Dapp y todas las opciones que ofrece. Alguna funcionalidad irá acompañada de imagenes explicitas de su funcionamiento y además, se expondrán los diferentes errores que se controlan.

Si se está interesado en probar el funcionamiento de la aplicación, aquí se deja el repositorio de GitHub donde ha sido desarrollada:

https://github.com/TFG-DAOs/developer-community

8.2.1. Inicio

En esta pantalla inicial se encuentran las diferentes funcionalidades que ofrece el proyecto, como por ejemplo, añadir un nuevo perfil (New Profile), añadir una transición (Add Transition) y añadir un miembro (Add Member). También se puede observar que hay un perfil *Anonymous* por defecto, que no tiene transiciones a otros perfiles y tampoco miembros de este perfil.

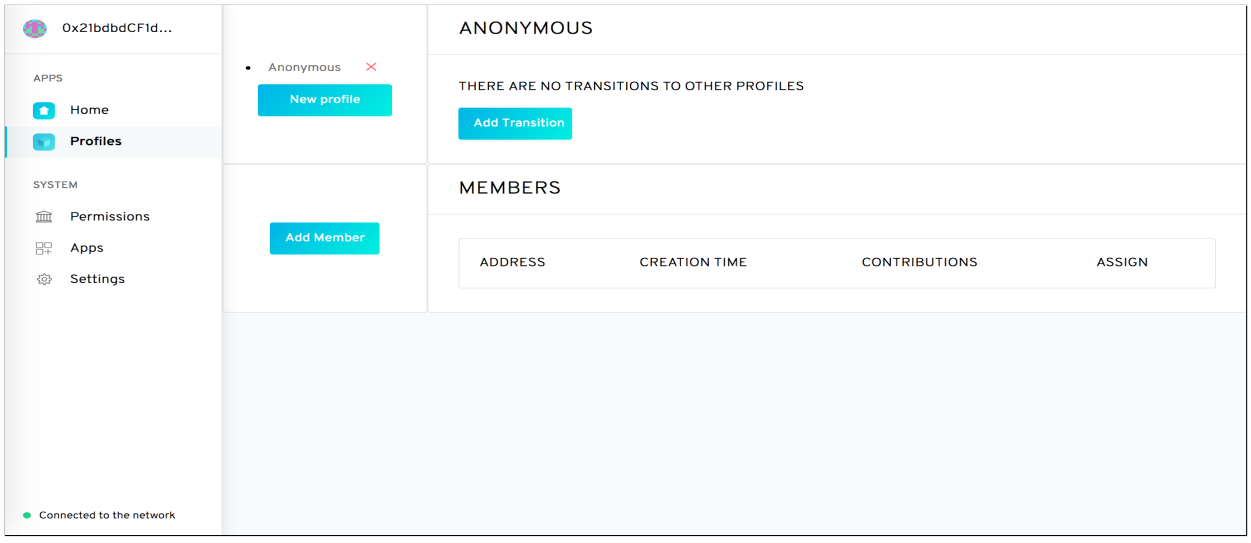


Figura 8.1: *Pantalla principal de la aplicación*

8.2.2. Añadir Perfil

En esta imagen se puede ver que ocurre cuando se pincha en el botón “New Profile”. Aparece un “SidePanel” del lateral con un campo input en el cual se introduce el nombre del nuevo perfil a añadir, en este caso en concreto sería “Support”.

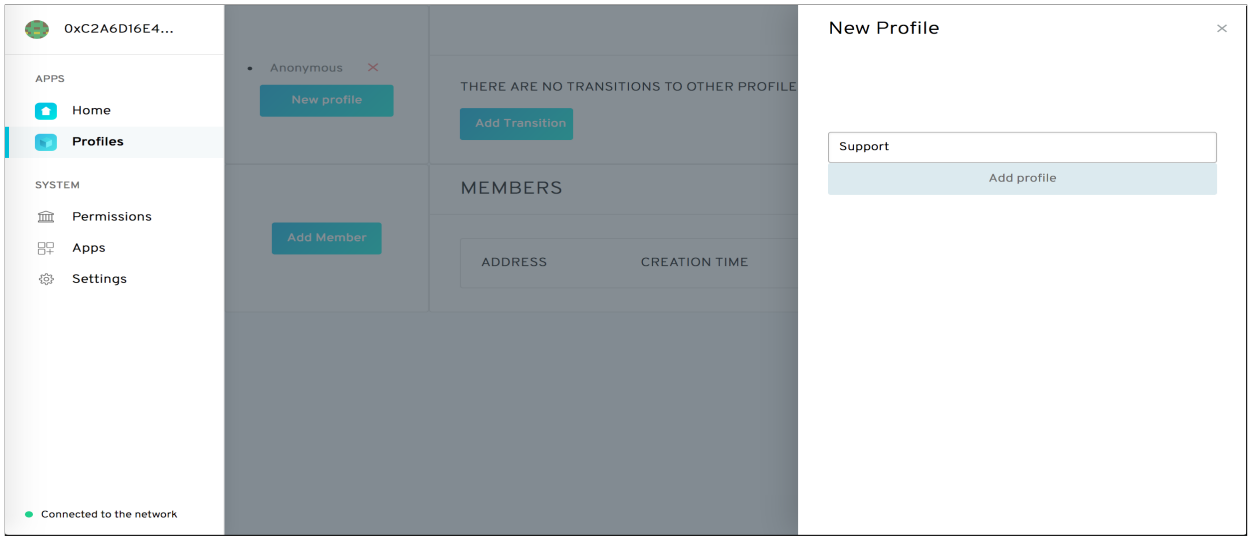


Figura 8.2: *Pantalla de Add Profile*

Al realizar esta acción se comprueba si el perfil ya existe, y en ese caso, no se permite hacer la transacción. Metamask avisa de que no se va producir, pero si se quiere saber, de manera más concreta, cual es el error, al hacer clic en “Confirmar” aparecerá lo siguiente:

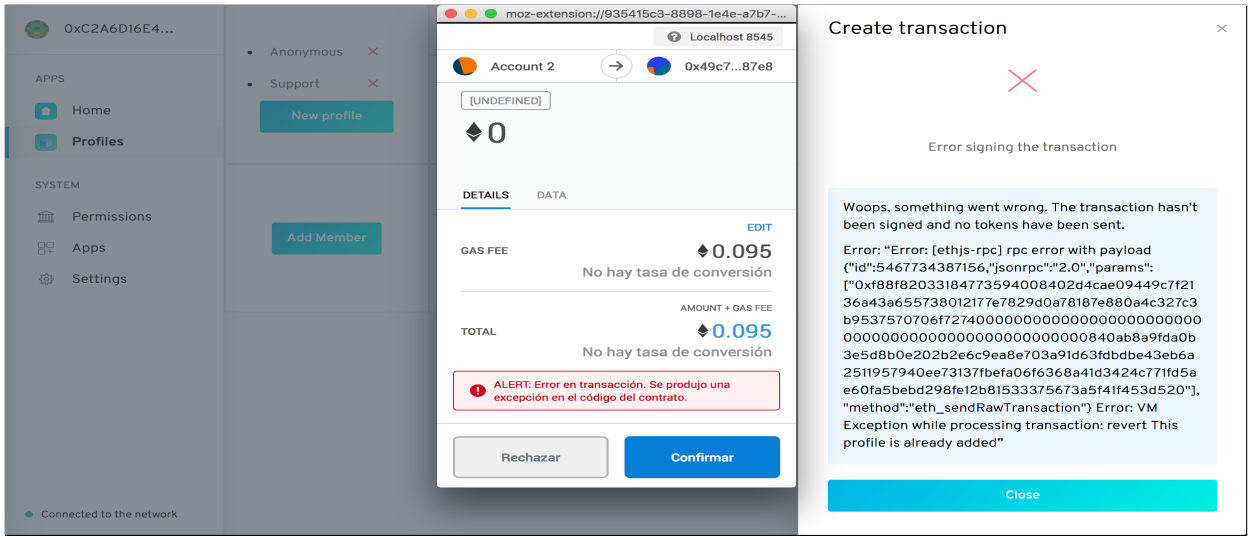


Figura 8.3: *Pantalla de Add Profile. Error: Perfil ya existe*

8.2.3. Añadir Transición

A continuación se va a añadir una transición. Para ello se selecciona el origen de la transición en la lista de perfiles, en este caso "Anonymous", y se hace clic en el botón "Add Transition".

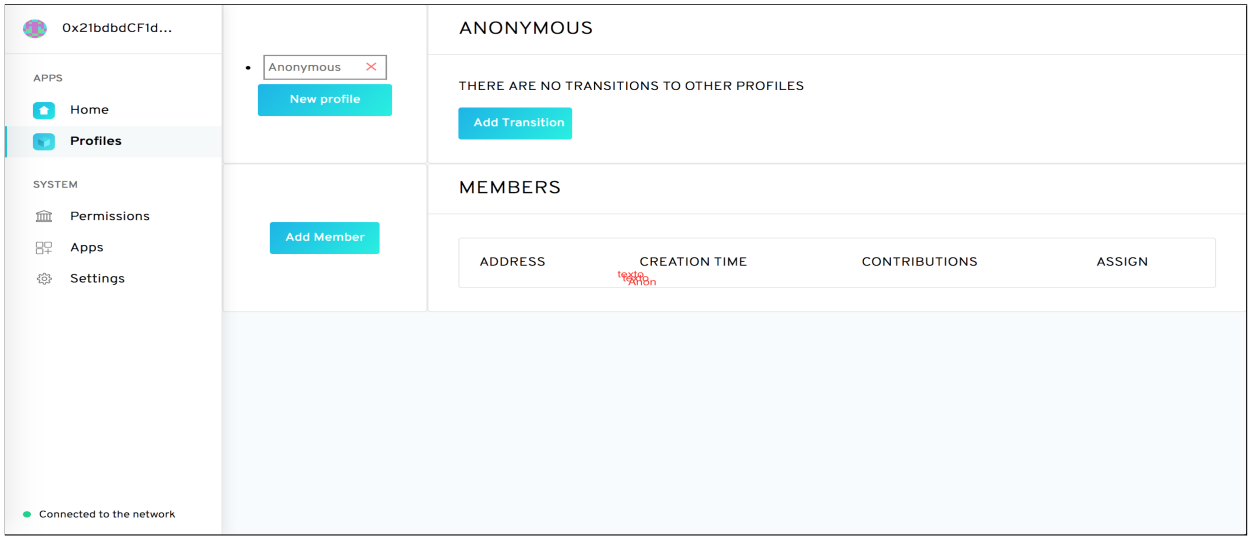


Figura 8.4: *Pantalla de Home. Anonymous seleccionado*

Aparecería un "SidePanel" del latera en el cual se podrán añadir dos condiciones que son AND entre ellas : el tiempo que hay que permanecer en el perfil de origen y el número de contribuciones que son necesarias para transitar al perfil final. Por último se elegirá en el "dropdown" el perfil final de la transición y se hará clic en el botón "Add".

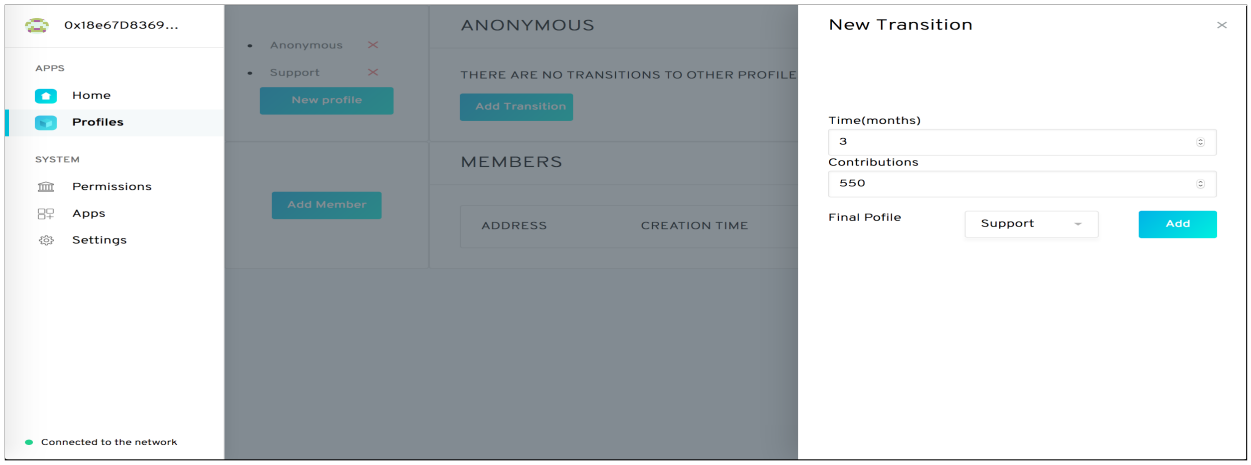


Figura 8.5: *Pantalla de Add Transition*

Una vez añadida la transición, lo que ocurrirá es que se añadirá una nueva fila en la tabla de transiciones, con los datos que se le han indicado anteriormente, añadiendo dos botones al final. El de "Settings" que te permite modificar la transición y la "X" que te permite borrar la transición.

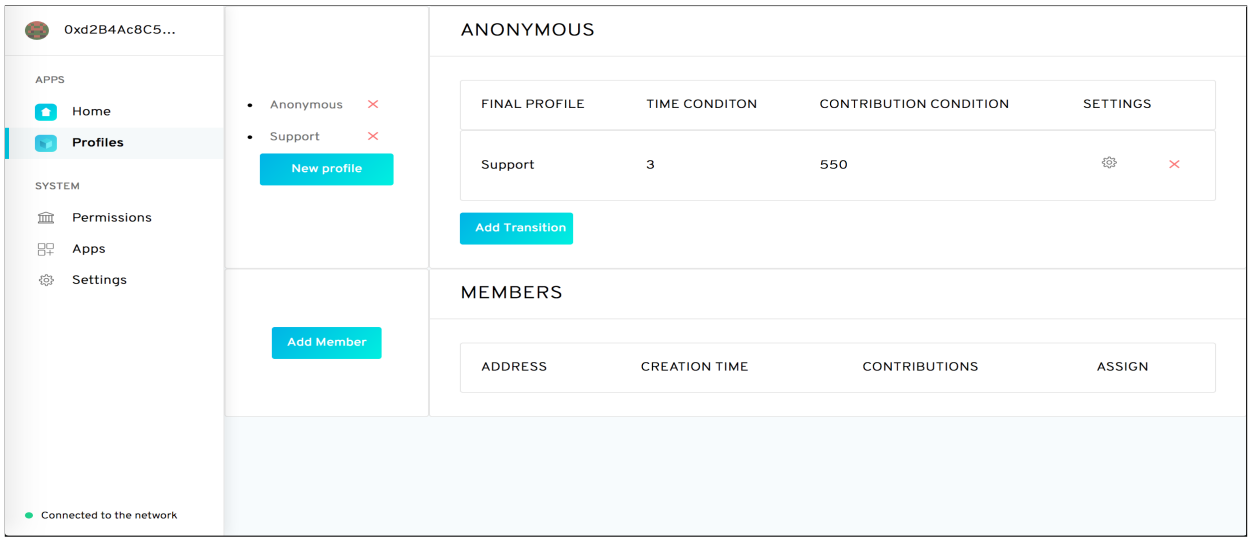


Figura 8.6: *Pantalla de Add Transition. Éxito: Transición añadida*

Durante el proceso de añadir una transición, se pueden producir una serie de errores los cuales están controlados. Los errores serían:

- Intentar añadir una transición de un perfil que no exista a uno que si exista o viceversa
- La transición no tenga las contribuciones o los meses con valor negativo.

8.2.4. Eliminar Perfil

Un perfil puede ser eliminado siempre y cuando no tenga ninguna transición asociada a él. Para borrar un perfil, lo único que hay que hacer es clicar en la "X" que aparece en la figura 8.6 al lado del nombre de "Support". Una vez clicado aparecerá el siguiente "SidePanel" preguntando si se está seguro de querer borrar el perfil "Support".

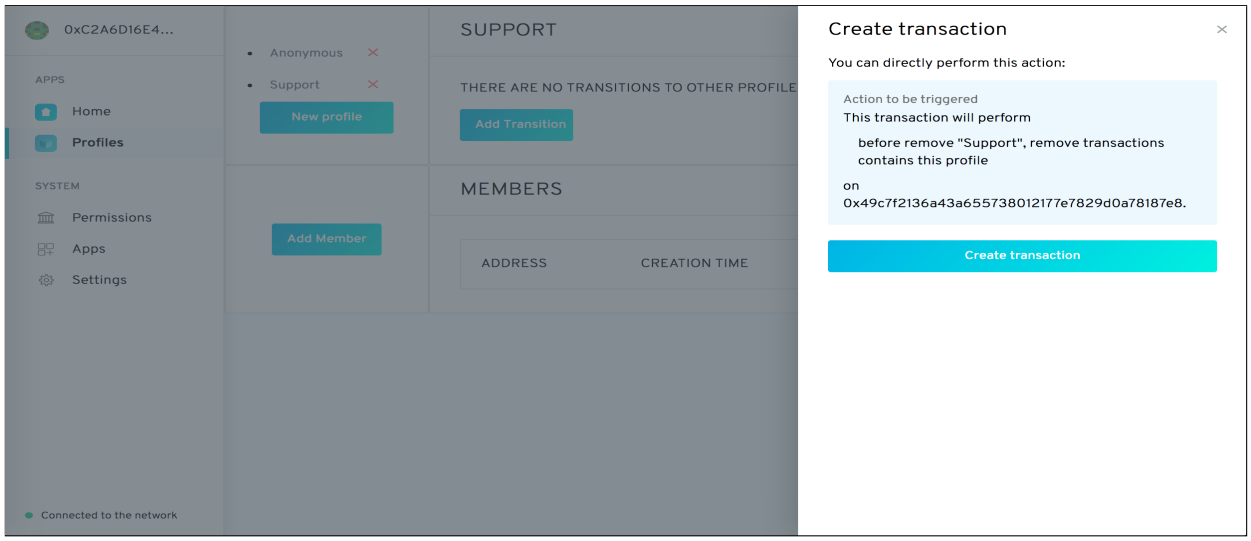


Figura 8.7: *Pantalla de Remove Profile*

En caso afirmativo se clicaría en "Create transition" y se eliminaría. Por último, si el perfil tuviese una transición asociada aparecería el siguiente mensaje de error:

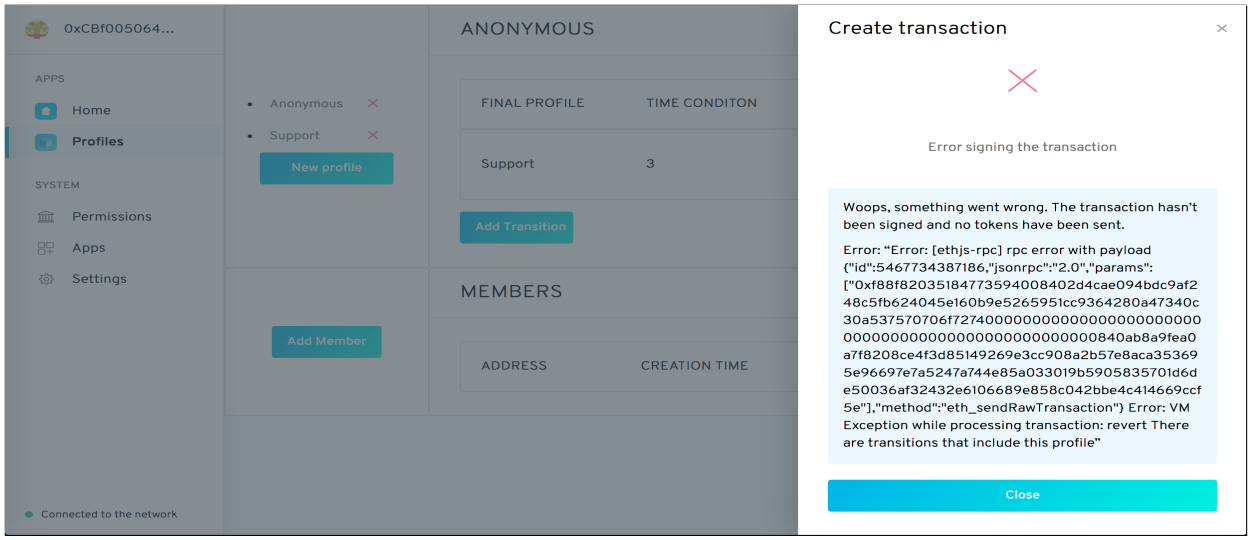


Figura 8.8: *Pantalla de Remove Profile. Error: Perfil con transición asociada*

8.2.5. Eliminar Transición

Si lo que se quiere es eliminar una transición, lo único que habría que hacer sería pulsar en la "X" al lado del engranaje de "Settings".

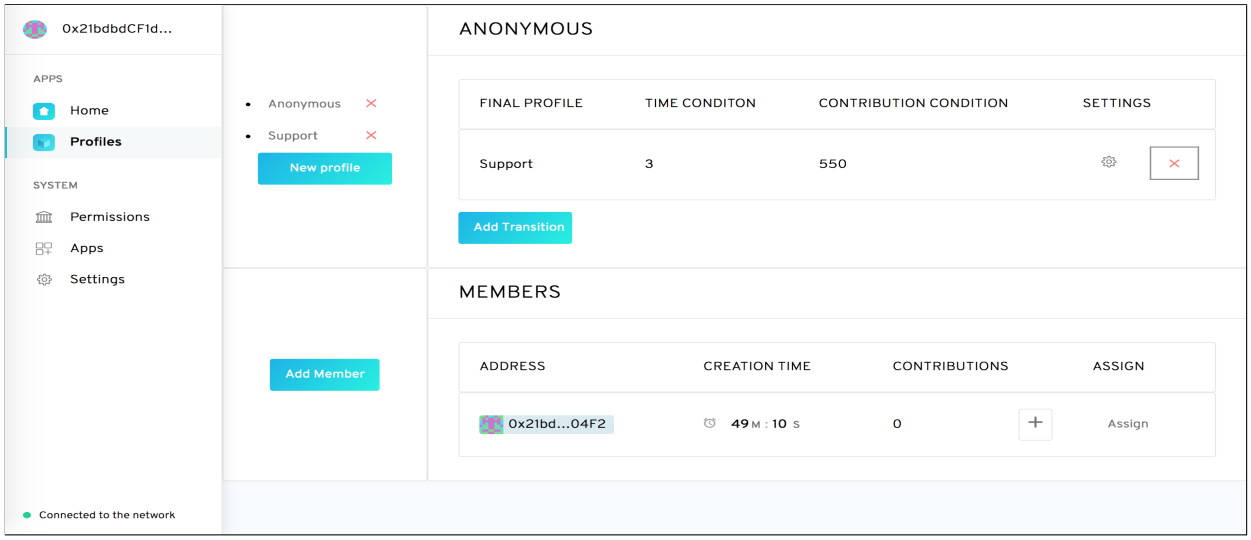


Figura 8.9: *Pantalla de Remove Transition. Botón "X" marcado*

Y aparecería un "SidePanel" avisando de que se va a eliminar la transición.

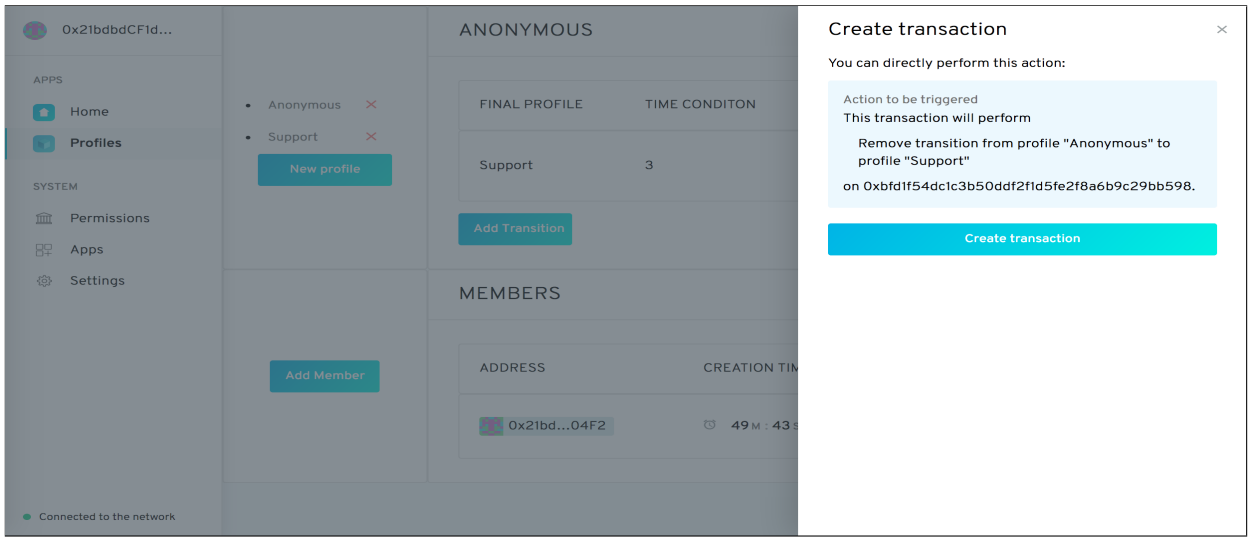


Figura 8.10: *Pantalla de Remove Transition. Éxito: mensaje de aviso*

8.2.6. Modificar Transición

Cambiar las condiciones de una transición se puede realizar solo si existen transiciones, ya que si no tiene transiciones no hay nada que modificar, ni aparece el botón de "Settings" en la tabla que vemos a continuación:

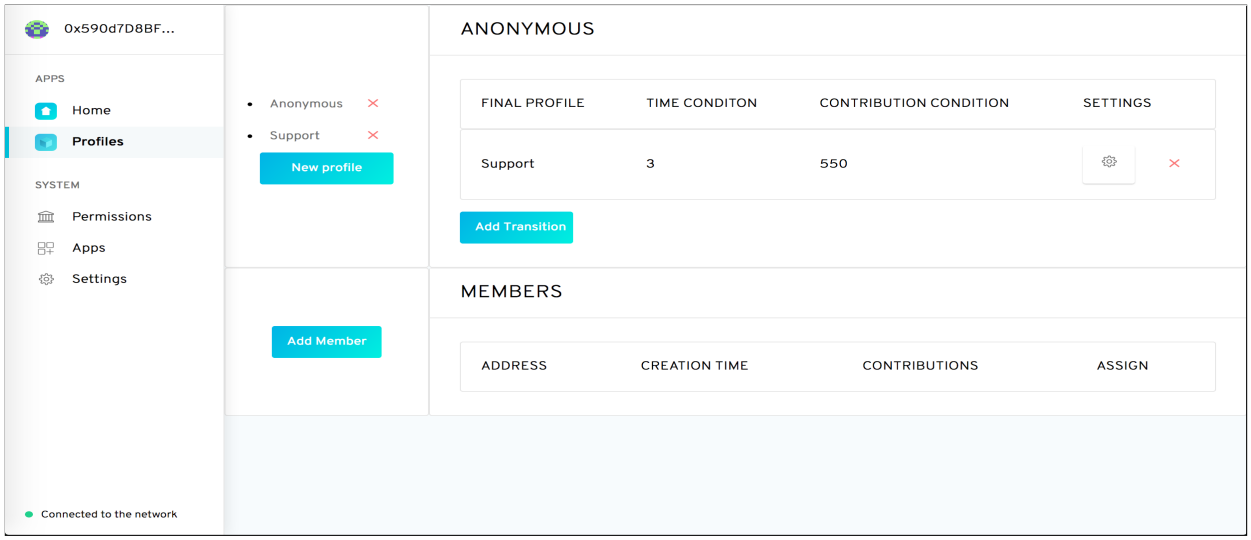


Figura 8.11: *Pantalla de Change Conditions. Éxito: Condiciones cambiadas*

En la siguiente imagen aparece el "SidePanel" que se despliega una vez pulsamos el botón de "Settings" de una transición, donde aparecen los campos "Time" y "Contributions", los cuales se pueden modificar según las necesidades de la transición.

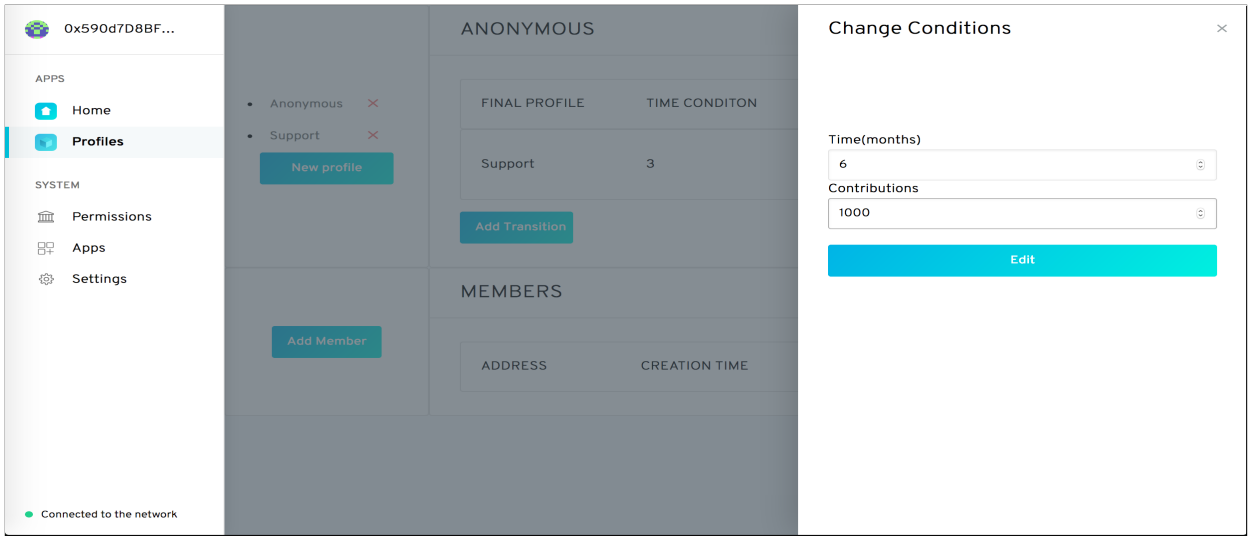


Figura 8.12: *Pantalla de Change Conditions. Éxito: Condiciones cambiadas*

Los principales errores de esta Función son las características de las condiciones, es decir, el tiempo tiene que ser positivo y mayor que cero, en cambio las contribuciones pueden ser mayores o iguales a cero. Por lo que si algún campo de los que se van a modificar es un valor de los mencionados anteriormente se producirían errores.

8.2.7. Añadir Miembro

En la imagen se observa que existe el botón de "Add Member". Al hacer clic en dicho botón, aparecerá un "SidePanel" con un input donde introduciremos la dirección de 32 bytes del miembro que queremos añadir.

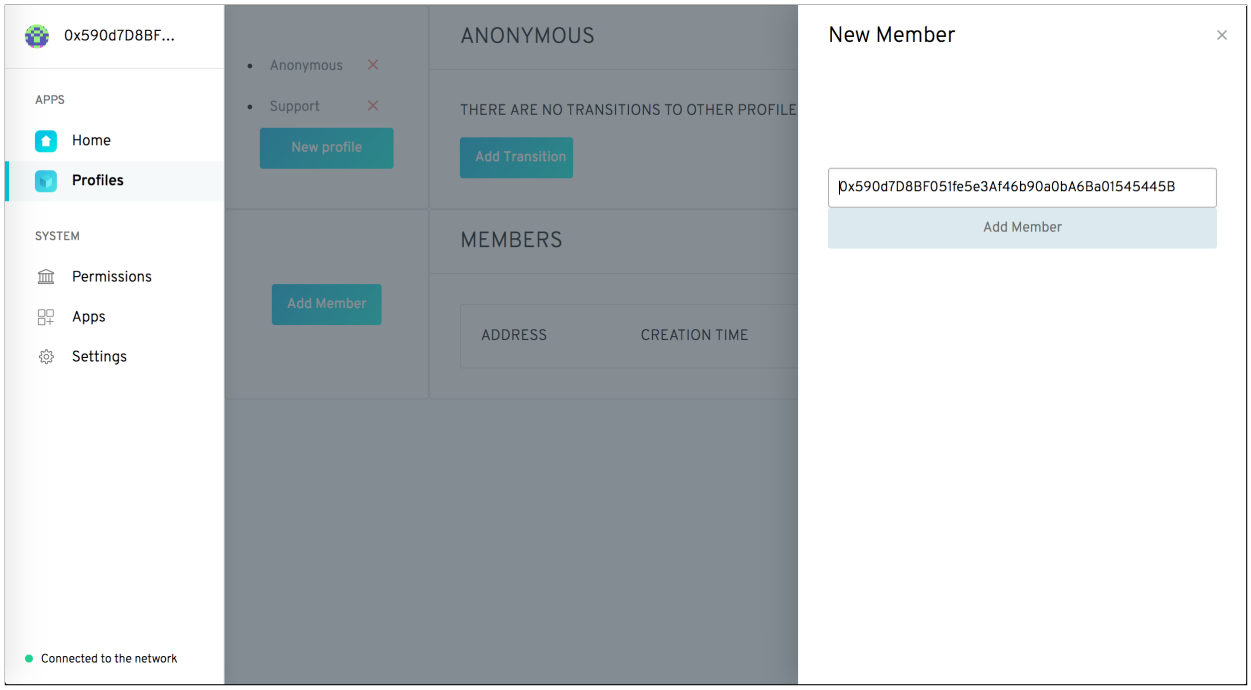


Figura 8.13: *Pantalla de Add Member*

Después se hará clic en el boton "Add Member" del *SidePanel* y aparecerá el siguiente mensaje de confirmación:

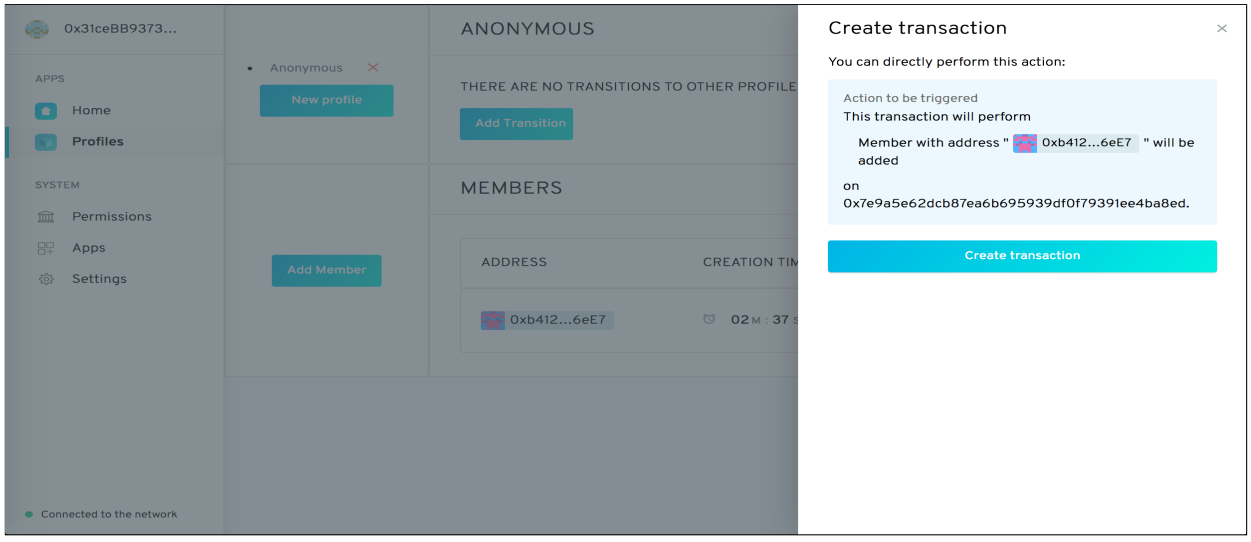


Figura 8.14: *Pantalla de Add Member*

Una vez se clique en el botón de "Create Transaction", el miembro se añadirá con el perfil

por defecto “Anonymous”.

En el transcurso de añadir un miembro se pueden producir el error de que el usuario ya exista donde recibirás el siguiente mensaje de error:

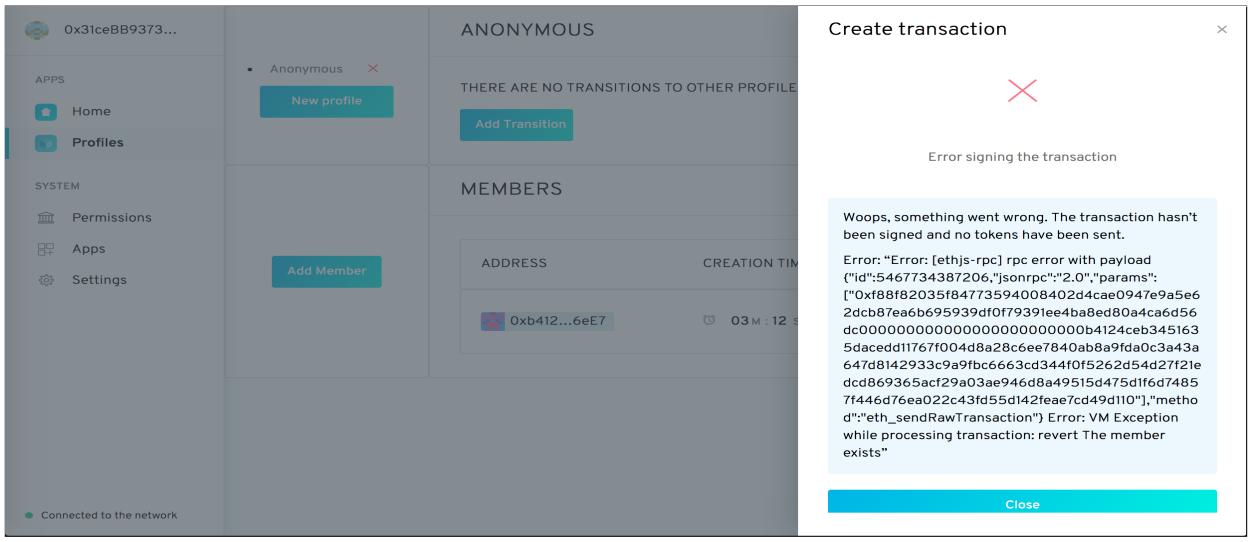


Figura 8.15: *Pantalla de Add Member. Error: El miembro ya existe*

8.2.8. Asignar Perfil a Miembro

Una vez se crea un miembro, en la tabla de miembros existe la opción "Assign" que te permite asignar un perfil a un miembro

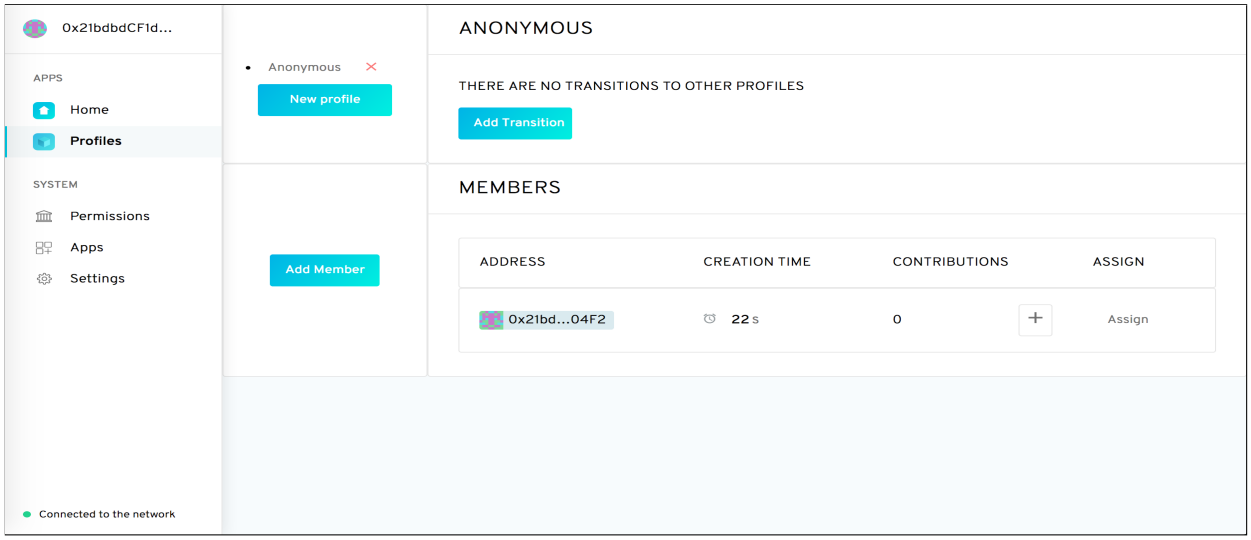


Figura 8.16: *Pantalla de Add Member*

Pero, este botón de "Assign" solo te permitirá asignar un perfil nuevo al miembro en caso de que este cumpla con todas las condiciones necesarias para poder transitar del perfil en el que se encuentra a otro perfil. En caso de que no pueda, le saldrá el siguiente mensaje de información:

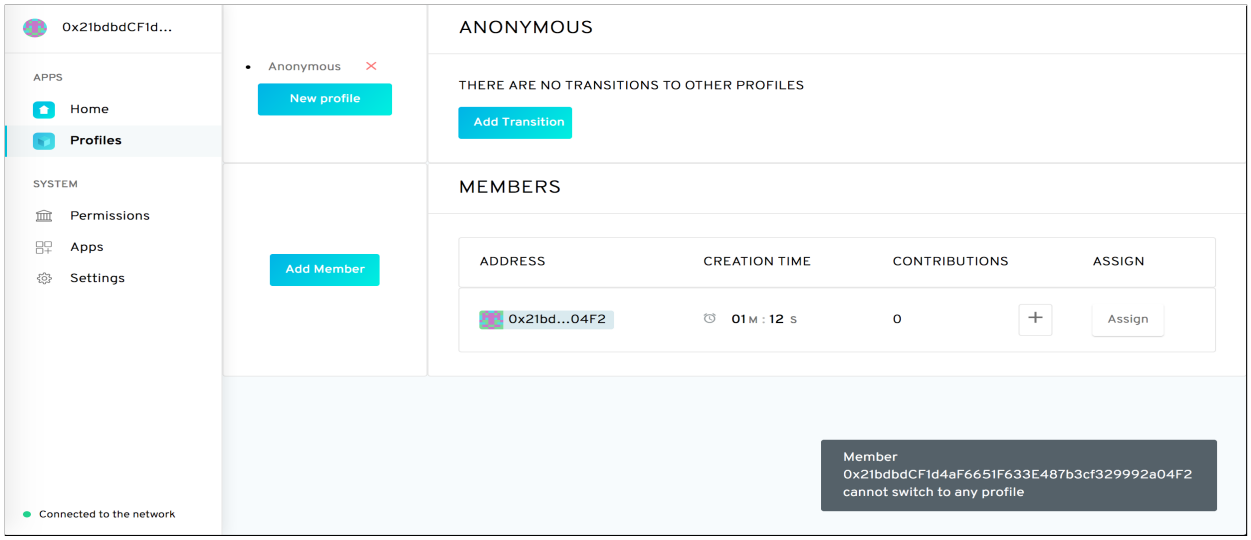


Figura 8.17: *Pantalla de Assign Member. Error: No cumple las condiciones para hacer la transición a ningún perfil*

En caso de que el miembro cumpla los criterios de alguna transición, el botón de "Assign" aparecerá rodeado por una linea gris. Se hará clic en él y aparecerá un "SidePanel" con un "dropdown" el cual contiene los perfiles a los que el miembro puede hacer la transición.

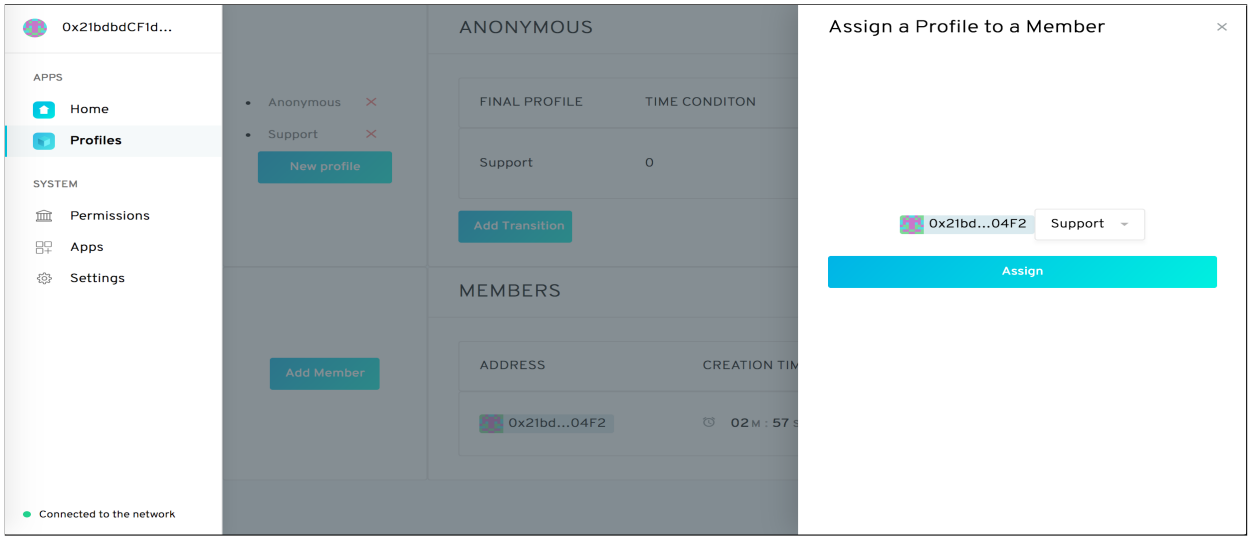


Figura 8.18: *Pantalla de Assign Member. Éxito: combobox con los perfiles a los que puede hacer la transición*

8.2.9. Incrementar contador de contribuciones

Cuando se quiere incrementar el número de contribuciones de un miembro, se hará click en el boton "+" de la columna "Contributions".

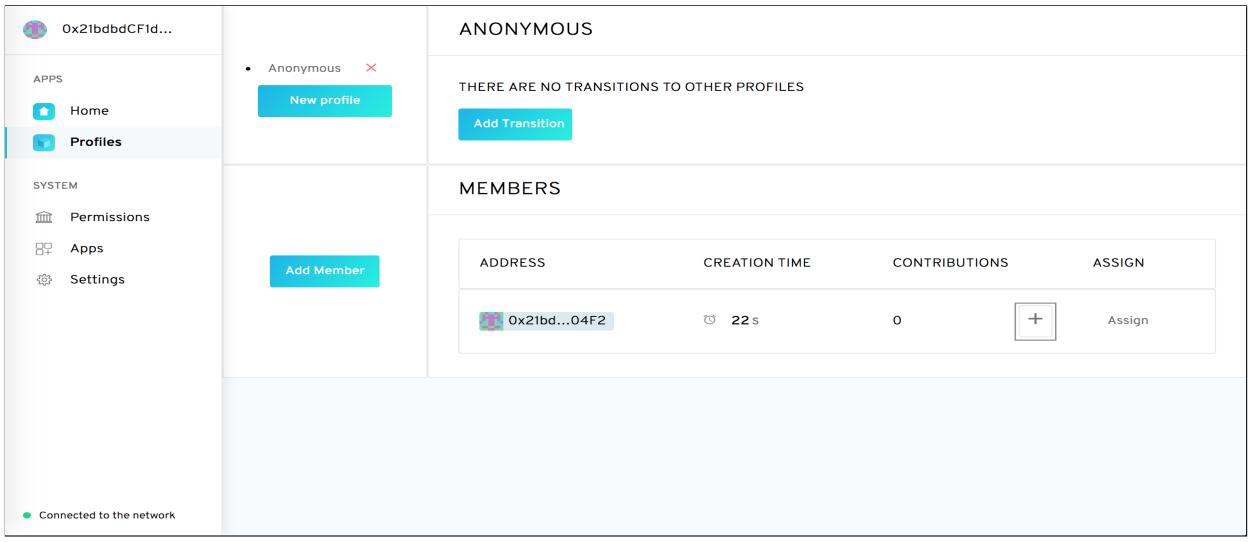


Figura 8.19: *Pantalla de Home con el botón "+" marcado*

Una vez pulsado el botón el siguiente mensaje indicando que se va a incrementar en uno la contribuciones del miembro:

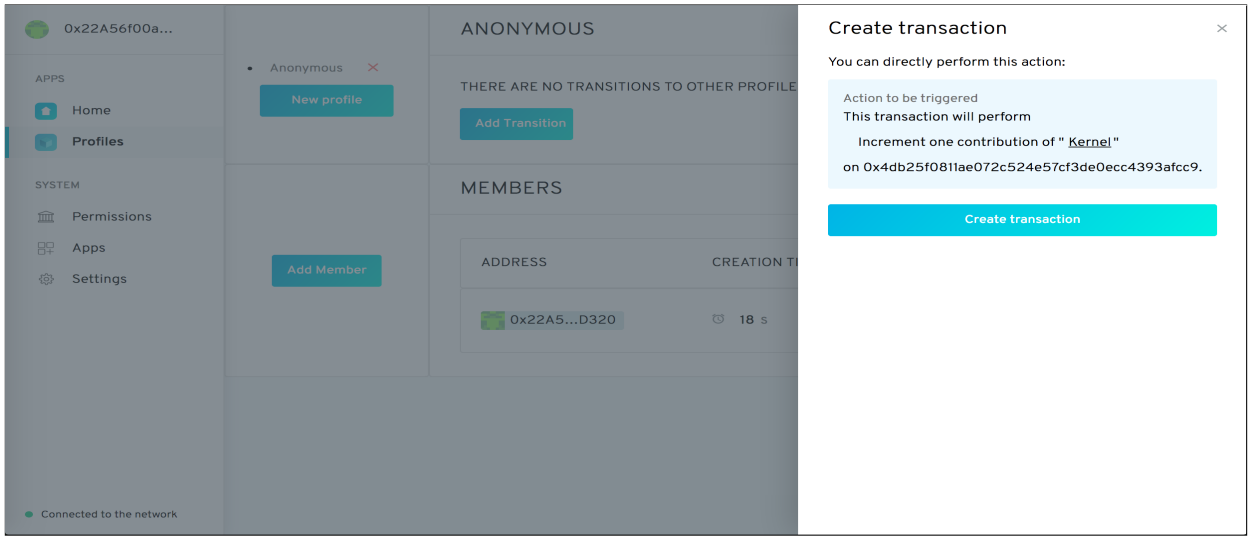


Figura 8.20: *Pantalla de Increment Contribution*

8.2.10. Asignar Rol a Miembro

Los roles serían características o capacidades que se le pueden asignar a los miembros. En este caso se ha añadido el rol de poder aumentar las contribuciones del resto de miembros. Como se puede ver en la siguiente imagen, el miembro "Kernel" tiene el rol asignado de incrementar contribuciones:

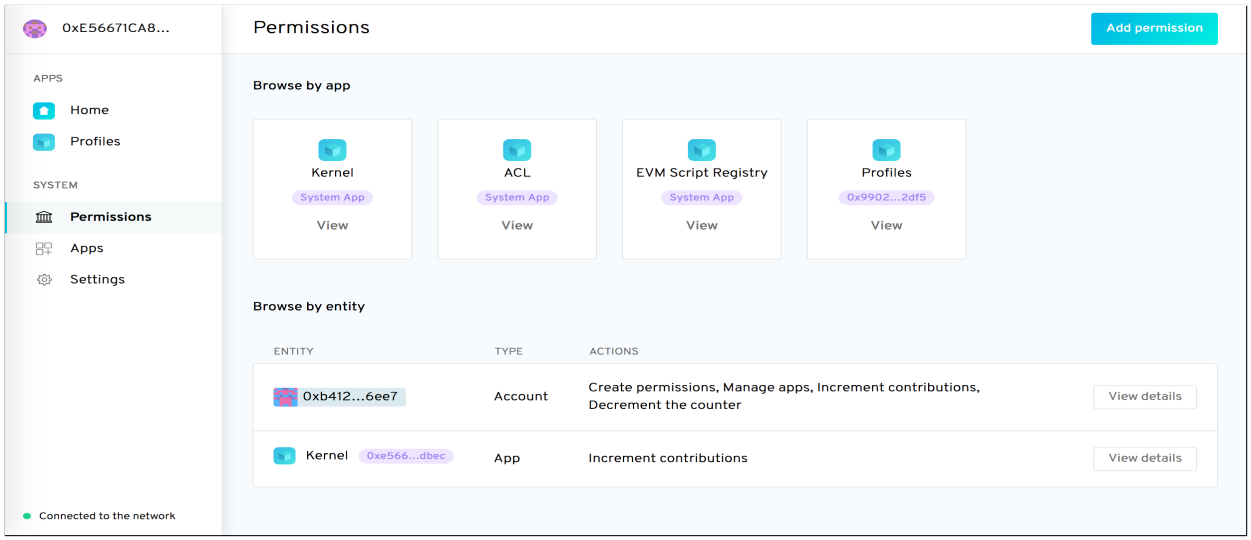


Figura 8.21: *Pantalla para ver los Roles de los Miembros*

En caso de que estés en la cuenta de un miembro que no tenga el rol "Increment contributions" asignado, aparecería el siguiente mensaje de error al intentar añadir una contribución a un miembro:

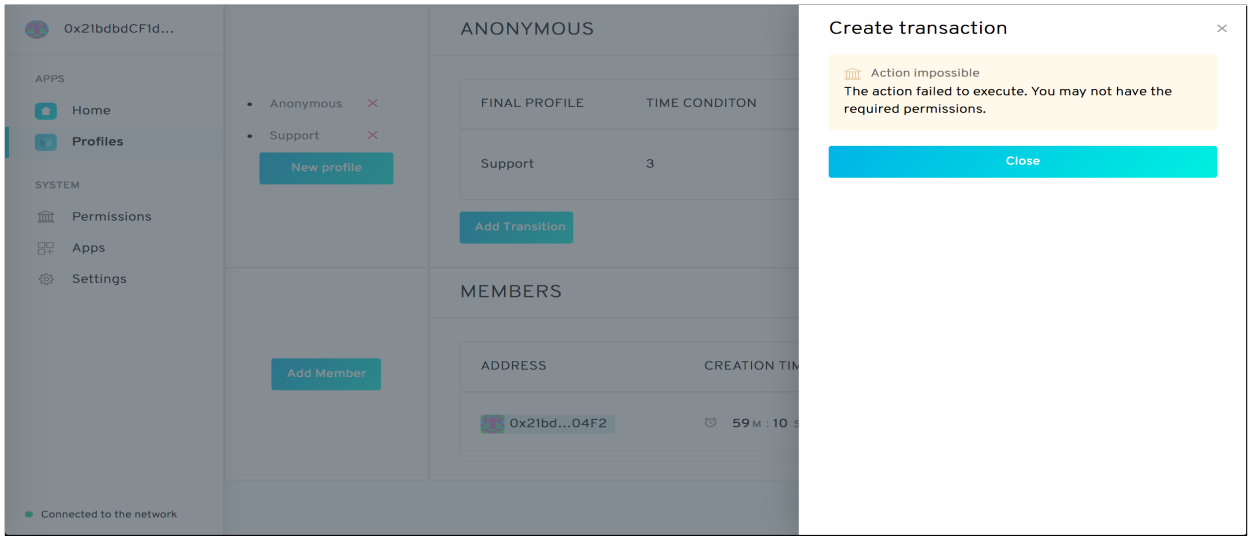


Figura 8.22: *Pantalla de error por no tener el rol de "Increment contributions" asignado*

8.2.11. Implementación de Tests

Para terminar, falta por explicar la implementación de los test. Como su propio nombre indica, son una batería de pruebas que se realizan al *smart contract* para comprobar que funciona correctamente. Los tests nos permiten poder ejecutar diferente casuísticas dentro del contrato y comprobar que se desenvuelve correctamente en todo tipo de situaciones.

```
Starting ganache-cli...
Running ganache-cli with pid 92230 in port 8545
Running tests...
npx: instaló 54 en 2.909s
Using network 'rpc'.


  Contract: ProfileManager
    ✓ should be add a Profile (152ms)
    ✓ should be exists a default "Anonymous" profile (87ms)
    ✓ should be remove "Support" profile (196ms)


  3 passing (4s)
```

Figura 8.23: *Consola con ejecución de tests*

En este caso en concreto se han implementado tres tests:

- "should be add a Profile": comprueba que se puede añadir un perfil correctamente.
- "should be exists a default "Anonymous" profile": comprueba si existe el perfil "Anonymous" por defecto.
- "should be remove "Support" profile": añade un perfil "Support" y comprueba que se ha eliminado correctamente.

Capítulo 9

Conclusiones y trabajo futuro

Los objetivos de este trabajo eran **probar y comparar los *frameworks*** disponibles para poder crear DAOs y, además **crear un modelo de DAO** que sirviera como guía para cualquier tipo de aplicación y organización. Estos objetivos los hemos **conseguido** a través de la realización de nuestra app **"Guerrilla Developers"** ya que la hemos desarrollado sobre Aragon después de analizar otros dos *frameworks*, además las funciones del *smart contract* pueden usarse para cualquier estructura de una organización.

Después de analizar los diferentes *frameworks* podemos concluir que la creación de **DAOs** cada vez **tiene más apoyo** a través de distintas **plataformas** aunque al tratarse de una tecnología que esta empezando ahora todavía tiene que madurar. Durante los meses de trabajo hemos visto como plataformas como Colony o DAOstack que al principio desechamos porque tenían poco desarrollo **han avanzado en poco tiempo** hasta convertirse en plataformas a tener en cuenta junto con Aragon.

Con la realización de esta aplicación nos hemos dado cuenta de las **múltiples posibilidades** que existen para que cualquier organización pueda **crear una DAO** siempre que tenga conocimientos mínimos. La gestión de las DAOs se puede aplicar desde una comunidad de vecinos, por ejemplo, realizando votaciones para la toma de decisiones en base a las propuestas de los vecinos para mejorar la comunidad, hasta una empresa de desarrollo de software como pudiera ser "Guerrilla Developer".

Nuestra app abre la puerta a la **creación de DAOs en los que existe una gestión de miembros** ya que el **modelo** que hemos creado está **pensado** para **todo tipo de jerarquías** y **todo tipo de funciones** dentro de las organizaciones. La creación de este modelo permite que se pueda **avanzar mucho más rápido** en la creación y los desarrolladores se puedan centrar más en las especificaciones de la organización en sí, dejando en segundo plano las funciones y funcionalidades básicas.

Uno de los **problemas** que hemos identificado es que el *framework* está en **continuo desarrollo** (actualización cada trimestre) y por tanto las **DAOs** que se creen, deben estar también en **continua actualización**, ya que muchas estructuras y formas de hacer cambian. Por otra parte, hemos llegado a la conclusión de que los **usuarios deben tener** al menos ciertos **conocimientos sobre blockchain** y las tecnologías que se utilizan en el *framework* para poder utilizarlas adecuadamente.

Trabajo futuro

En cuanto al trabajo futuro, se podría seguir desarrollando "Guerrilla Developers" **automatizando** la asignación de roles a perfiles, es decir, sin acceder al menú del sistema y asignarlos manualmente, los cuales podrán realizar acciones en función del perfil que contengan, por ejemplo, realizar peticiones de reforma de la DAO mediante votaciones a las cuales solo podrían acceder ciertos perfiles.

También se podría mejorar el sistema de transiciones cambiando las contribuciones existentes por **contabilizar lineas de código**, para que a su vez sea más exacto el número de contribuciones de cada miembro y así el reparto de fondos sea más equitativo, para ello podríamos **sincronizar nuestra DAO con Github** y sea capaz de contabilizar las que programa cada miembro.

Por último, otra de las posibles actualizaciones de nuestra DAO, sería la de incluir un

sistema de **reparto automático de fondos**, en función del trabajo desarrollado, así como la **creación de un token** de reputación que te permita tener más peso en la toma de decisiones y también que este token pueda ser añadido en las condiciones de las transacciones.

Capítulo 10

Conclusions and future work

The objective of this work was **to test and compare the available frameworks** in order to create DAOs and also to **create a DAO model** that would serve as a guide for any type of application and organization. These objectives have been achieved through the realization of our app **"Guerrilla Developers"** because, we have developed on Aragon after analyzing two other frameworks and also the functions of the smart contract can be used for any structure of an organization.

After analyzing the different frameworks we can conclude that the creation of **DAOs is increasingly supported through** different **platforms** but as it is a technology that is starting now has yet to mature. During the months of work we have seen how platforms like Colony or DAOstack that at first we discarded because they had **little development have advanced** to become platforms to be considered together with Aragon.

With the realization of this application we have realized the **many possibilities** that exist for any organization to **create a DAO** as long as it has minimal knowledge. The management of the DAOs can be applied from a community of neighbors, for example, voting for decision making based on the proposals of neighbors to improve the community, to a software development company such as "Guerrilla Developer".

Our app opens the door to **the creation of DAOs in which there is member mana-**

gement as the **model** we have created **is designed for all kinds of hierarchies and** all kinds of **functions** within organizations. The creation of this model makes it possible **to advance much faster** in the creation and the developers can focus more on the specifications of the organization itself leaving in second place the basic functions and functionalities.

One of the **problems** we have identified is that the framework is in **continuous development** (update every quarter) and therefore the DAOs that are created must also be in continuous update, because many structures and ways of doing things change. On the other hand, we have come to the conclusion that **users must have** at least some **knowledge about blockchain** and the technologies that are used in the framework to be able to use them properly.

Future work

As for the future work, "Guerrilla Developers" could continue to be developed by assigning roles to profiles **automatically**, that is, without accessing the system menu and assigning them manually, which may perform actions depending on the profile they have, for example, make requests for reform of the DAO through votes to which only certain profiles could access.

Also the system of transitions could be improved by changing the existing contributions **to count lines of code**, so that in turn the number of contributions of each member is more exact and thus the distribution of funds is more equitable, for this we could **synchronize our DAO with Github** and be able to count the ones programmed by each member. On the other hand, as it is a generic DAO of transition of profiles it can be applied to any organization if it is approached in an appropriate way.

Finally, another possible update of our DAO would be to include a system of **automatic distribution of funds**, depending on the work developed, as well as **the creation of a**

token of reputation that allows you to have more weight in decision-making and also, that this token can be added in the conditions of transactions.

Capítulo 11

Bibliografía

[1] Blockchain: https://en.wikipedia.org/wiki/Blockchain

[2] PRENAFETA RODRÍGUEZ, J., "Smart contracts: aproximación al concepto y problemática legal básica", en Diario La Ley, núm. 8824, 15.11.2016 (LA LEY 6489/2016).

[3]DAO: https://blockchainhub.net/dao-decentralized-autonomous-organization/

[4] DAOstack: https://daostack.io/

[5] Colony: https://colony.io/

[6] Metamask: https://metamask.io/

[7] NodeJs: https://nodejs.org/

[8] NPM: https://docs.npmjs.com/

[9] Solidity: https://solidity.readthedocs.io/

[10] ES6 : https://labs.beeva.com/esto-es-lo-que-deberías-saber-de-es6-el-nuevo-javascript-8fc5ee031a5c

[11][12]"Learning React: Functional Web Development with React adn Redux" Edicion: 1ª. Editorial: O'Reilly. Autor: Alex Bancks y Eve Porcello. Año 2017.

[13] Web3: https://web3js.readthedocs.io/en/1.0/

[14] Aragon: https://wiki.aragon.org/

[15] DAO: https://es.cointelegraph.com/ethereum-for-beginners/what-is-dao

[16] Economía colaborativa: https://economipedia.com/definiciones/economia-colaborativa.html

[17] Economia colaborativa: https://www.elsaltodiario.com/economia/duenos-economia-colaborativa-uber-airbnb

[18] DAO: https://bitcoin.es/criptomonedas/que-es-un-dao-organizacion-autonoma-descentralizada/

[19] Desarrollo App: https://www.udemy.com/ethereum-and-solidity-the-complete-developers-guide/learn/v4/overview

[20] Desarrollo App: https://www.udemy.com/react-fundamentals/learn/v4/overview

[21] AGPLv3: https://opensource.org/licenses/AGPL-3.0

[22] MakerDAO: https://makerdao.com/es/

[23] Digix : https://digix.global/dgd/

[24]The Decentralized Autonomous Organization Social Science Research Network (SSRN). Governance Issues. Consultado el 5 de diciembre de 2017 (ingles).

[25] Parity Frozen Funds: https://es.cointelegraph.com/news/parity-multisig-wallet-hacked-or-how-come

[26]"The DAO": https://www.xataka.com/seguridad/the-dao-y-el-caso-del-robo-de-los-50-millones-de-dolares-en-ethereum-insert-coin-1x01